# Programming online tasks in barebones JavaScript

Thomas E. Gladwin

## Contents



# Introduction

This is a tutorial on programming a psychological reaction-time task in (barebones) JavaScript that participants can complete online in their browser. I'm assuming you can make simple HTML pages and have done some basic JavaScript tutorials. Also, be aware I'm a researcher who programs, not a proper software developer!

The idea is to provide a general purpose “programming pattern” which can easily be adjusted to particular tasks.

(Brief note to pre-empt a possible source of confusion: JavaScript is not the same programming language as Java. JavaScript is a very high-level language, i.e., the technical details are hidden, that lives in every browser - you can run a line of JavaScript right now if you know the way to the Web Console in your browser. Java is a more low-level language that you need to install specially and that has to be compiled into programs that have to be run by a special Java program.)

The first steps require nothing but a browser. Later steps will require you to either install PHP on your computer, or work online via your own website. If you want to launch a study online you'll need hosting in any case (I recommend DreamHost). But let's look at the JavaScript basics first.

Essential: Program along with the tutorial. The idea is that throughout every step you build up your program by adding and adjusting it based on each new listing. It'll take some attention

and effort! In some cases I've emphasized new code, but please always just run though the new listings and compare them with your previous step.

# Step 1: Hello World

First we're going to make what's basically the introduction screen of the task.

First we're going to need a web page in which the JavaScript task will be embedded. This will require some boilerplate: you need to load your JavaScript task when the webpage loads, and you need a HTML "Canvas" object to draw on. Let's make a webpage called task.html (this is a simple ASCII text file you can open in any notepad program) with that Canvas first:

```
<html>
<head>
	<title>My Task</title>
</head>

<body>
	<canvas id = "mycanvas" style = "background: black; display:block; margin: 0px auto;"
width = "600" height = "600"></canvas>

</body>
</html>
```

To test (always be testing), open the local file in a browser. It should have the title "My Task" and should contain a black square. The black square is where the magic will happen. It's essential that it has the correct ID, here id = "mycanvas", since the JavaScript uses the ID to access the canvas.

Now we need to tell the page to load the JavaScript file we'll be making. The file doesn't exist yet, but let's get the page ready for it as below:

```
<html>
<head>
	<title>My Task</title>
	<script src = "task_code.js"></script>
</head>

<body onload="init()">
	<canvas id = "mycanvas" style = "background: black; display:block; margin: 0px auto;"
width = "600" height = "600"></canvas>

	<script>
		function init() {
			myTask = new Task();
			myTask.init();
		};
	</script>
</body>
</html>
```

Note that you added two script sections and an onload function to the body tag. The script tags tell the browser that you're adding some JavaScript code. The first snippet tells the browser to load the JavaScript file called task_code.js (which we'll be making next). The file will contain the task. The second snippet defines a function called init, which will set up and start the task which you loaded previously. The onload function in the body tag will run this init function as soon as the page is loaded (at which point the JavaScript file will have been loaded and ready for use).

Now make a text file task_code.js containing this code, which defines a Task object:

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		this.startTask();
	};

	this.startTask = function() {
		this.cx.fillStyle = "green";
		this.cx.font = "32px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		this.cx.fillText("Hello World!", 300, 300);
	}

}
```

If you need to debug, use the Browser Console and the command console.log("my debug info here") at strategic places in your code. The "use strict" a the top of the code helps to prevent some nasty bugs by turning them into errors.

Test it. Hopefully you've now seen your first step into online task programming!

Now we're obviously going to want to make a general show-text function so we can recycle rather than commit the sin of copy-pasting code:

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		this.startTask();
	};
```

```
	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
	}

	this.startTask = function() {
		this.showText("Hello World!", 0, 0);
	}

}
```

The helper function showText also lets you give x and y coordinates on a -1 to 1 scale, with (0, 0) being the centre of the screen, instead of having to think about specific pixel positions.

That's it for step 1!

# Step 2: Trials and Blocks

Now we're going to make a trial within which some events happen. This all uses one particular trick: functions linked by timers. JavaScript, unlike some specialized task programming languages, doesn't work like a sequence of commands being executed in the order they're written. By nature, it's asynchronous - it's designed to respond to things happening on a webpage. We can control timing users timing events as in the following code. I've also added a clearCanvas function, since by default writing text to screen doesn't delete what was already on the canvas.

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		this.startTask();
	};

	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
```

```
	}

	this.emptyCanvas = function() {
		this.cx.fillStyle = "black";
		this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
	};

	this.startTask = function() {
		this.showText("Hello World!", 0, 0);
		setTimeout(this.trial_init.bind(this), 1000);
	}

	// Trials

	this.trial_init = function() {
		this.trial_fixation();
	}

	this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 1000);
	}

	this.trial_stimulus = function() {
		this.emptyCanvas();
		this.showText(":-)", 0, 0);
		setTimeout(this.trial_ITI.bind(this), 1000);
	}

	this.trial_ITI = function() {
		this.emptyCanvas();
		setTimeout(this.trial_init.bind(this), 1000);
	}

}
```

So the trial is initialized by the task init() function (nothing much happening here yet); this links to the fixation cross; this links to the stimulus; and this links to the InterTrial Interval; and this loops back to the trial init. Each link is created by the setTimeout function, which activates a function after the specified duration. The bind(this) is essential to make sure JavaScript still knows what the relevant "this" object is at the point the function gets activated - by default, this won't be the task object, and things won't work.

This is clearly getting close to a task, and for some purposes you might pretty much have the basics already. Let's add a block loop around the trials, so there are nBlock blocks each containing nTrial trials. But remember JavaScript is asynchronous, so it's not going to look like a usual for-loop - it's going to use the pattern of linked functions and timers, with a condition at the end of a sequence telling us whether to exit the loop or run it again. Make sure you recognize this loop-until linked timers pattern, since it's the main structure of the task code.

```
"use strict";
```

```
var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		// Task parameters
		this.nBlocks = 3;
		this.nTrials = 5;
		this.currentBlock = 0;
		this.currentTrial = 0;

		this.startTask();
	};

	// Helper functions

	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
	}

	this.emptyCanvas = function() {
		this.cx.fillStyle = "black";
		this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
	};

	// Task start

	this.startTask = function() {
		this.showText("Hello World!", 0, 0);
		setTimeout(this.block_init.bind(this), 1000);
	}

	// Trials

	this.trial_init = function() {
		this.trial_fixation();
	}

	this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
	}

	this.trial_stimulus = function() {
		this.emptyCanvas();
		this.showText(":-) Trial " + this.currentTrial, 0, 0);
		setTimeout(this.trial_ITI.bind(this), 500);
	}
```

```
	this.trial_ITI = function() {
		this.emptyCanvas();
		setTimeout(this.trial_end.bind(this), 250);
	}

	this.trial_end = function() {
		this.currentTrial++;
		if (this.currentTrial >= this.nTrials) {
			this.block_end();
		} else {
			this.trial_init();
		}
	}

	// Blocks
	this.block_init = function() {
		this.currentTrial = 0;

		this.emptyCanvas();
		this.showText("Block " + (this.currentBlock + 1) + " starting...", 0, 0);

		setTimeout(this.trial_init.bind(this), 1000);
	}

	this.block_end = function() {
		this.currentBlock++;
		if (this.currentBlock >= this.nBlocks) {
			this.task_end();
		} else {
			this.block_init();
		}
	}

	// End of task
	this.task_end = function() {
		this.showText("End", 0, 0);
	}
}
```

Look at it go! Note the changes in the code, e.g., that the relevant task parameters and loop control variables have now been added to the init function, and that the task start now links to the block init function, instead of the trial init function. Note that the magic happens in the trial_end and block_end functions. Also note that I'm following the convention that in code, counting starts from zero, not from one.

# Step 3: Button presses

Now to add responses. The trick here - again, this is the asynchronous style, unlike a "wait_for_response" type command in a more linear language - is to add a kind of daemon that listens for a response, all the time. When the daemon hears something, it activates a function which we can then use for the task. Let's make the task wait for a button press before starting. We'll need one more control variable for this to tell the response function whether it's the right time to do something. (If you want a quick extra test, put an alert("Pressed") function in the handler.)

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		// Task parameters
		this.nBlocks = 3;
		this.nTrials = 5;
		this.currentBlock = 0;
		this.currentTrial = 0;

		// Response handler flags
		this.waiting_for_task_start = 0;

		this.startTask();
	};

	// Helper functions

	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
	}

	this.emptyCanvas = function() {
		this.cx.fillStyle = "black";
		this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
	};

	// Task start

	this.startTask = function() {
		this.showText("Hello World!", 0, 0);
		this.showText("Press a key to start", 0, 0.5);
		this.waiting_for_task_start = 1;
	}

	// Trials

	this.trial_init = function() {
		this.trial_fixation();
	}

	this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
	}
```

```
	this.trial_stimulus = function() {
		this.emptyCanvas();
		this.showText(":-) Trial " + this.currentTrial, 0, 0);
		setTimeout(this.trial_ITI.bind(this), 500);
	}

	this.trial_ITI = function() {
		this.emptyCanvas();
		setTimeout(this.trial_end.bind(this), 250);
	}

	this.trial_end = function() {
		this.currentTrial++;
		if (this.currentTrial >= this.nTrials) {
			this.block_end();
		} else {
			this.trial_init();
		}
	}

	// Blocks
	this.block_init = function() {
		this.currentTrial = 0;

		this.emptyCanvas();
		this.showText("Block " + (this.currentBlock + 1) + " starting...", 0, 0);

		setTimeout(this.trial_init.bind(this), 1000);
	}

	this.block_end = function() {
		this.currentBlock++;
		if (this.currentBlock >= this.nBlocks) {
			this.task_end();
		} else {
			this.block_init();
		}
	}

	// End of task
	this.task_end = function() {
		this.showText("End", 0, 0);
	}

	// Response handling
	this.respHandler = function(event) {
		event.preventDefault();
		if (this.waiting_for_task_start == 1) {
			this.waiting_for_task_start = 0;
			this.block_init();
		}
	};

	window.addEventListener("keydown", this.respHandler.bind(this));

}
```

Note that the startTask function doesn't now link to the block start, but just sets the flag that we're now waiting for the task to start. This is now moved to the response handler, which will start the block but only if that flag is up. Make sure to lower the flag (set it back to 0) when the desired action is taken, so it doesn't keep happening!

Let's add a button-press to make blocks start, using the same pattern. Now we'll use a flag waiting_for_block_start, raised in block init instead of linking through to the trials. Again, that's done by the response handler, when it's activated by a button press and only if the flag is raised.

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		// Task parameters
		this.nBlocks = 3;
		this.nTrials = 5;
		this.currentBlock = 0;
		this.currentTrial = 0;

		// Response handler flags
		this.waiting_for_task_start = 0;
		this.waiting_for_block_start = 0;

		this.startTask();
	};

	// Helper functions

	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
	}

	this.emptyCanvas = function() {
		this.cx.fillStyle = "black";
		this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
	};

	// Task start

	this.startTask = function() {
		this.showText("Hello World!", 0, 0);
		this.showText("Press a key to start", 0, 0.5);
		this.waiting_for_task_start = 1;
```

```
}

// Trials

this.trial_init = function() {
	this.trial_fixation();
}

this.trial_fixation = function() {
	this.emptyCanvas();
	this.showText("+", 0, 0);
	setTimeout(this.trial_stimulus.bind(this), 250);
}

this.trial_stimulus = function() {
	this.emptyCanvas();
	this.showText(":-) Trial " + this.currentTrial, 0, 0);
	setTimeout(this.trial_ITI.bind(this), 500);
}

this.trial_ITI = function() {
	this.emptyCanvas();
	setTimeout(this.trial_end.bind(this), 250);
}

this.trial_end = function() {
	this.currentTrial++;
	if (this.currentTrial >= this.nTrials) {
		this.block_end();
	} else {
		this.trial_init();
	}
}

// Blocks
this.block_init = function() {
	this.currentTrial = 0;

	this.emptyCanvas();
	this.showText("Block " + (this.currentBlock + 1), 0, 0);
	this.showText("Press a key to start", 0, 0.5);

	this.waiting_for_block_start = 1;
}

this.block_end = function() {
	this.currentBlock++;
	if (this.currentBlock >= this.nBlocks) {
		this.task_end();
	} else {
		this.block_init();
	}
}

// End of task
this.task_end = function() {
	this.showText("End", 0, 0);
}

// Response handling
```

```
	this.respHandler = function(event) {
		event.preventDefault();
		if (this.waiting_for_task_start == 1) {
			this.waiting_for_task_start = 0;
			this.block_init();
		} else if (this.waiting_for_block_start == 1) {
			this.waiting_for_block_start = 0;
			this.trial_init();
		}
	};

	window.addEventListener("keydown", this.respHandler.bind(this));

}
```

And there we have the backbone of the block-trial looping structure and the presentation of trial events!

# Step 4: RT measurement

So now we can respond to responses, we want to collect RTs. To do that, we'll use a stopwatch system: we'll store the time at which the stimulus was presented on screen (which happens after the relevant function ends), and store the time a response came in, and take the difference. Check where time_stimulus and time_response are give a value by the Date.now() function, which gives a time in milliseconds (as it happens, the time is given in milliseconds since Jan 1 1970, but we're only interested in differences). There's now also an RT variable to be filled in - initialized per trial at zero, and given a stopwatch-based value by the response handler if a response is given. Again, there's flag for this, since we don't want unexpected effects.

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		// Task parameters
		this.nBlocks = 3;
		this.nTrials = 5;
		this.currentBlock = 0;
		this.currentTrial = 0;

		// Response handler flags
		this.waiting_for_task_start = 0;
		this.waiting_for_block_start = 0;
		this.waiting_for_trial_response = 0;

		// Timing variables
		this.time_stimulus = 0;
		this.time_response = 0;

		// Performance variables
```

```
		this.RT = 0;

		this.startTask();
};

// Helper functions

this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
}

this.emptyCanvas = function() {
		this.cx.fillStyle = "black";
		this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
};

// Task start

this.startTask = function() {
		this.showText("Hello World!", 0, 0);
		this.showText("Press a key to start", 0, 0.5);
		this.waiting_for_task_start = 1;
}

// Trials

this.trial_init = function() {
		this.RT = 0;
		this.trial_fixation();
}

this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
}

this.trial_stimulus = function() {
		this.emptyCanvas();
		this.showText("Trial " + this.currentTrial, 0, -0.5);
		this.showText("Press a key!", 0, 0);
		setTimeout(this.trial_ITI.bind(this), 1000);
		this.waiting_for_trial_response = 1;
		this.time_stimulus = Date.now();
}

this.trial_ITI = function() {
		this.emptyCanvas();
		this.showText("RT was " + this.RT, 0, 0);
		setTimeout(this.trial_end.bind(this), 250);
}
```

```
	this.trial_end = function() {
		this.currentTrial++;
		if (this.currentTrial >= this.nTrials) {
			this.block_end();
		} else {
			this.trial_init();
		}
	}

	// Blocks
	this.block_init = function() {
		this.currentTrial = 0;

		this.emptyCanvas();
		this.showText("Block " + (this.currentBlock + 1), 0, 0);
		this.showText("Press a key to start", 0, 0.5);

		this.waiting_for_block_start = 1;
	}

	this.block_end = function() {
		this.currentBlock++;
		if (this.currentBlock >= this.nBlocks) {
			this.task_end();
		} else {
			this.block_init();
		}
	}

	// End of task
	this.task_end = function() {
		this.showText("End", 0, 0);
	}

	// Response handling
	this.respHandler = function(event) {
		this.time_response = Date.now();
		event.preventDefault();
		if (this.waiting_for_task_start == 1) {
			this.waiting_for_task_start = 0;
			this.block_init();
		} else if (this.waiting_for_block_start == 1) {
			this.waiting_for_block_start = 0;
			this.trial_init();
		} else if (this.waiting_for_trial_response == 1) {
			this.waiting_for_trial_response = 0;
			this.RT = this.time_response - this.time_stimulus;
		}
	};

	window.addEventListener("keydown", this.respHandler.bind(this));

}
```

Now to finish off, let's make this into a task that requires particular responses to particular stimuli: When you see a ":-)" press F, when you see a ":-(" press J.

We'll now specify an array of response keys via the capital letter of the keys we want to use. The associated numerical character codes, which used by the response handler, are then determined by the init function. The response handler takes the information in the response event generated by a key press and checks which index in the response key array it was. Here, F is index 0 and J is index 1. A -1 means none of the specified response keys were pressed. Firstly, we'll just make sure we're getting the correct response information. Note that we initialize the new variables to default values when initializing each trial.

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		// Task parameters
		this.nBlocks = 3;
		this.nTrials = 5;
		this.currentBlock = 0;
		this.currentTrial = 0;

		// Response handler flags
		this.waiting_for_task_start = 0;
		this.waiting_for_block_start = 0;
		this.waiting_for_trial_response = 0;

		// Timing variables
		this.time_stimulus = 0;
		this.time_response = 0;

		// Performance variables
		this.RT = 0;
		this.respKeys = ["F", "J"];
		this.respCodes = [];
		for (var i = 0; i < this.respKeys.length; i++) {
			this.respCodes.push(this.respKeys[i].charCodeAt(0));
		}
		this.pressedRespKeyIndex = -1;
		this.pressedRespKey = "";
		this.acc = 0;

		this.startTask();
	};

	// Helper functions

	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
		this.cx.fillText(text0, x_screen, y_screen);
```

```
}

this.emptyCanvas = function() {
	this.cx.fillStyle = "black";
	this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
};

// Task start

this.startTask = function() {
	this.showText("Smiley and Frownies", 0, 0);
	this.showText("Smiley: Press F", 0, 0.2);
	this.showText("Frownie: Press J", 0, 0.4);
	this.showText("Press a key to start", 0, 0.8);
	this.waiting_for_task_start = 1;
}

// Trials

this.trial_init = function() {
	this.RT = 0;
	this.acc = 0;
	this.pressedResponseKeyIndex = -1;
	this.pressedRespKey = "None";
	this.trial_fixation();
}

this.trial_fixation = function() {
	this.emptyCanvas();
	this.showText("+", 0, 0);
	setTimeout(this.trial_stimulus.bind(this), 250);
}

this.trial_stimulus = function() {
	this.emptyCanvas();
	this.showText("Trial " + this.currentTrial, 0, -0.5);
	this.showText("Press a key!", 0, 0);
	setTimeout(this.trial_ITI.bind(this), 1000);
	this.waiting_for_trial_response = 1;
	this.time_stimulus = Date.now();
}

this.trial_ITI = function() {
	this.emptyCanvas();
	this.showText("RT was " + this.RT, 0, 0);
	this.showText("Response key was " + this.pressedRespKey, 0, 0.5);
	setTimeout(this.trial_end.bind(this), 1000);
}

this.trial_end = function() {
	this.currentTrial++;
	if (this.currentTrial >= this.nTrials) {
		this.block_end();
	} else {
		this.trial_init();
	}
}

// Blocks
this.block_init = function() {
```

```
		this.currentTrial = 0;

		this.emptyCanvas();
		this.showText("Block " + (this.currentBlock + 1), 0, 0);
		this.showText("Press a key to start", 0, 0.5);

		this.waiting_for_block_start = 1;
	}

	this.block_end = function() {
		this.currentBlock++;
		if (this.currentBlock >= this.nBlocks) {
			this.task_end();
		} else {
			this.block_init();
		}
	}

	// End of task
	this.task_end = function() {
		this.showText("End", 0, 0);
	}

	// Response handling
	this.respHandler = function(event) {
		this.time_response = Date.now();
		event.preventDefault();
		if (this.waiting_for_task_start == 1) {
			this.waiting_for_task_start = 0;
			this.block_init();
		} else if (this.waiting_for_block_start == 1) {
			this.waiting_for_block_start = 0;
			this.trial_init();
		} else if (this.waiting_for_trial_response == 1) {
			this.waiting_for_trial_response = 0;
			// Get RT
			this.RT = this.time_response - this.time_stimulus;
			// Get pressed response key, as an index in the respKey array
			this.pressedRespKey = "Illegal key";
			for (var i = 0; i < this.respCodes.length; i++) {
				if (event.keyCode == this.respCodes[i]) {
					this.pressedRespKeyIndex = i;
					this.pressedRespKey =
this.respKeys[this.pressedRespKeyIndex];
				}
			}
		}
	};

	window.addEventListener("keydown", this.respHandler.bind(this));

}
```

Now let's randomly select one of the two stimuli to present per trial; check accuracy; and give accuracy feedback, for which we'll insert a new trial event before the ITI.

```
"use strict";

var Task = function() {
```

```
this.init = function() {
	// Initialize the variables to draw on the canvas
	this.mycanvas = document.querySelector("#this.mycanvas");
	this.cx = this.mycanvas.getContext("2d");

	// Task parameters
	this.nBlocks = 3;
	this.nTrials = 5;
	this.currentBlock = 0;
	this.currentTrial = 0;

	// Response handler flags
	this.waiting_for_task_start = 0;
	this.waiting_for_block_start = 0;
	this.waiting_for_trial_response = 0;

	// Timing variables
	this.time_stimulus = 0;
	this.time_response = 0;

	// Performance variables
	this.RT = 0;
	this.respKeys = ["F", "J"];
	this.respCodes = [];
	for (var i = 0; i < this.respKeys.length; i++) {
		this.respCodes.push(this.respKeys[i].charCodeAt(0));
	}
	this.pressedRespKeyIndex = -1;
	this.pressedRespKey = "";
	this.acc = 0;

	// Stimuli
	this.stimuli = [":-)", ":-("];
	this.selectedStimulus = 0;

	this.startTask();
};

// Helper functions

this.showText = function(text0, x, y, color, fontsize) {
	if (color == undefined) {color = "white";}
	if (fontsize == undefined) {fontsize = 32;}
	this.cx.fillStyle = color;
	this.cx.font = fontsize + "px Georgia";
	this.cx.textAlign = "center";
	this.cx.textBaseline = "middle";
	let x_screen = (this.mycanvas.width / 2) * (1 + x);
	let y_screen = (this.mycanvas.height / 2) * (1 + y);
	this.cx.fillText(text0, x_screen, y_screen);
}

this.emptyCanvas = function() {
	this.cx.fillStyle = "black";
	this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
};

// Task start

this.startTask = function() {
```

```
		this.showText("Smiley and Frownies", 0, 0);
		this.showText("Smiley: Press F", 0, 0.2);
		this.showText("Frownie: Press J", 0, 0.4);
		this.showText("Press a key to start", 0, 0.8);
		this.waiting_for_task_start = 1;
}

// Trials

this.trial_init = function() {
		// Default values
		this.RT = 0;
		this.acc = 0;
		this.pressedResponseKeyIndex = -1;
		this.pressedRespKey = "None";
		// Randomize stimuli
		this.selectedStimulus = Math.floor(Math.random() * 2);
		// Start first trial
		this.trial_fixation();
}

this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
}

this.trial_stimulus = function() {
		this.emptyCanvas();
		let stimulusToShow = this.stimuli[this.selectedStimulus];
		this.showText(stimulusToShow, 0, 0);
		setTimeout(this.trial_feedback.bind(this), 1000);
		this.waiting_for_trial_response = 1;
		this.time_stimulus = Date.now();
}

this.trial_feedback = function() {
		this.emptyCanvas();
		if (this.pressedRespKeyIndex == this.selectedStimulus) {
			this.acc = 1;
		} else {
			this.acc = 0;
		}
		if (this.acc == 1) {
			this.showText("Correct!", 0, 0, "green");
		} else {
			this.showText("Incorrect!", 0, 0, "red");
		}
		setTimeout(this.trial_ITI.bind(this), 500);
}

this.trial_ITI = function() {
		setTimeout(this.trial_end.bind(this), 250);
}

this.trial_end = function() {
		this.currentTrial++;
		if (this.currentTrial >= this.nTrials) {
			this.block_end();
		} else {
```

```
                        this.trial_init();
                }
        }

        // Blocks
        this.block_init = function() {
                this.currentTrial = 0;

                this.emptyCanvas();
                this.showText("Block " + (this.currentBlock + 1), 0, 0);
                this.showText("Press a key to start", 0, 0.5);

                this.waiting_for_block_start = 1;
        }

        this.block_end = function() {
                this.currentBlock++;
                if (this.currentBlock >= this.nBlocks) {
                        this.task_end();
                } else {
                        this.block_init();
                }
        }

        // End of task
        this.task_end = function() {
                this.showText("End", 0, 0);
        }

        // Response handling
        this.respHandler = function(event) {
                this.time_response = Date.now();
                event.preventDefault();
                if (this.waiting_for_task_start == 1) {
                        this.waiting_for_task_start = 0;
                        this.block_init();
                } else if (this.waiting_for_block_start == 1) {
                        this.waiting_for_block_start = 0;
                        this.trial_init();
                } else if (this.waiting_for_trial_response == 1) {
                        this.waiting_for_trial_response = 0;
                        // Get RT
                        this.RT = this.time_response - this.time_stimulus;
                        // Get pressed response key, as an index in the respKey array
                        this.pressedRespKey = "Illegal key";
                        for (var i = 0; i < this.respCodes.length; i++) {
                                if (event.keyCode == this.respCodes[i]) {
                                        this.pressedRespKeyIndex = i;
                                        this.pressedRespKey =
this.respKeys[this.pressedRespKeyIndex];
                                }
                        }
                }
        };

        window.addEventListener("keydown", this.respHandler.bind(this));

}
```

Finally, let's end the response window immediately after a response is given. For this we need to store the timeOut object and destroy it in the response handler; and then run the next trial event function immediately from there.

```
"use strict";

var Task = function() {

	this.init = function() {
		// Initialize the variables to draw on the canvas
		this.mycanvas = document.querySelector("#this.mycanvas");
		this.cx = this.mycanvas.getContext("2d");

		// Task parameters
		this.nBlocks = 3;
		this.nTrials = 5;
		this.currentBlock = 0;
		this.currentTrial = 0;

		// Response handler flags
		this.waiting_for_task_start = 0;
		this.waiting_for_block_start = 0;
		this.waiting_for_trial_response = 0;

		// Timing variables
		this.currentTimeout = 0;
		this.time_stimulus = 0;
		this.time_response = 0;

		// Performance variables
		this.RT = 0;
		this.respKeys = ["F", "J"];
		this.respCodes = [];
		for (var i = 0; i < this.respKeys.length; i++) {
			this.respCodes.push(this.respKeys[i].charCodeAt(0));
		}
		this.pressedRespKeyIndex = -1;
		this.pressedRespKey = "";
		this.acc = 0;

		// Stimuli
		this.stimuli = [":-)", ":-("];
		this.selectedStimulus = 0;

		this.startTask();
	};

	// Helper functions

	this.showText = function(text0, x, y, color, fontsize) {
		if (color == undefined) {color = "white";}
		if (fontsize == undefined) {fontsize = 32;}
		this.cx.fillStyle = color;
		this.cx.font = fontsize + "px Georgia";
		this.cx.textAlign = "center";
		this.cx.textBaseline = "middle";
		let x_screen = (this.mycanvas.width / 2) * (1 + x);
		let y_screen = (this.mycanvas.height / 2) * (1 + y);
```

```
		this.cx.fillText(text0, x_screen, y_screen);
}

this.emptyCanvas = function() {
		this.cx.fillStyle = "black";
		this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
};

// Task start

this.startTask = function() {
		this.showText("Smiley and Frownies", 0, 0);
		this.showText("Smiley: Press F", 0, 0.2);
		this.showText("Frownie: Press J", 0, 0.4);
		this.showText("Press a key to start", 0, 0.8);
		this.waiting_for_task_start = 1;
}

// Trials

this.trial_init = function() {
		// Default values
		this.RT = 0;
		this.acc = 0;
		this.pressedResponseKeyIndex = -1;
		this.pressedRespKey = "None";
		// Randomize stimuli
		this.selectedStimulus = Math.floor(Math.random() * 2);
		// Start first trial
		this.trial_fixation();
}

this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
}

this.trial_stimulus = function() {
		this.emptyCanvas();
		let stimulusToShow = this.stimuli[this.selectedStimulus];
		this.showText(stimulusToShow, 0, 0);
		this.currentTimeout = setTimeout(this.trial_feedback.bind(this), 1000);
		this.waiting_for_trial_response = 1;
		this.time_stimulus = Date.now();
}

this.trial_feedback = function() {
		this.emptyCanvas();
		if (this.pressedRespKeyIndex == this.selectedStimulus) {
				this.acc = 1;
		} else {
				this.acc = 0;
		}
		if (this.acc == 1) {
				this.showText("Correct!", 0, 0, "green");
		} else {
				this.showText("Incorrect!", 0, 0, "red");
		}
		setTimeout(this.trial_ITI.bind(this), 500);
```

```
}

this.trial_ITI = function() {
	setTimeout(this.trial_end.bind(this), 250);
}

this.trial_end = function() {
	this.currentTrial++;
	if (this.currentTrial >= this.nTrials) {
		this.block_end();
	} else {
		this.trial_init();
	}
}

// Blocks
this.block_init = function() {
	this.currentTrial = 0;

	this.emptyCanvas();
	this.showText("Block " + (this.currentBlock + 1), 0, 0);
	this.showText("Press a key to start", 0, 0.5);

	this.waiting_for_block_start = 1;
}

this.block_end = function() {
	this.currentBlock++;
	if (this.currentBlock >= this.nBlocks) {
		this.task_end();
	} else {
		this.block_init();
	}
}

// End of task
this.task_end = function() {
	this.showText("End", 0, 0);
}

// Response handling
this.respHandler = function(event) {
	this.time_response = Date.now();
	event.preventDefault();
	if (this.waiting_for_task_start == 1) {
		this.waiting_for_task_start = 0;
		this.block_init();
	} else if (this.waiting_for_block_start == 1) {
		this.waiting_for_block_start = 0;
		this.trial_init();
	} else if (this.waiting_for_trial_response == 1) {
		this.waiting_for_trial_response = 0;
		// Get RT
		this.RT = this.time_response - this.time_stimulus;
		// Get pressed response key, as an index in the respKey array
		this.pressedRespKey = "Illegal key";
		for (var i = 0; i < this.respCodes.length; i++) {
			if (event.keyCode == this.respCodes[i]) {
				this.pressedRespKeyIndex = i;
```

```
                                                this.pressedRespKey =
this.respKeys[this.pressedRespKeyIndex];
                                        }
                                }
                                clearTimeout(this.currentTimeout);
                                this.trial_feedback();
                        }
                };

                window.addEventListener("keydown", this.respHandler.bind(this));

}
```

And so there we have (at least in terms of what the participant sees) a basic task!

# Step 5: Pictures

If you want to use images, the main technical trick is correctly loading the files. Presenting them is not much different than presenting text.

We're going to turn the two stimulus categories of smilies and frownies into two sets of images. Loading the images is, again, asynchronous - you don't load them in a loop one by one, but in a batch, and with a daemon to check when they're ready. Note the change to init and the new functions starting from init_stimuli(). You'll need two sets of four jpg files each, saved in subdirectories "Smilies" and "Frownies", within which the filenames are stim (1).jpg, stim (2).jpg, etc. In Windows, you can rename files like that all at once by selecting them all, pressing F2, and typing "stim".

Images are stored in a multidimensional array - one sub-array of images for each stimulus category. The recursive function createNextStim does the heavy lifting. The function loaded_a_stim, which createNextStim attaches to the daemons that are activated when images are successfully loaded, checks whether loading is complete, and then starts the task proper.

The images are shown using the new helder function showPicture(). Now in the initialization of each trial a random stimulus category is chosen, and a random stimulus from that category. The feedback function uses the new variable to determine accuracy.

```
"use strict";

var Task = function() {

        this.init = function() {
                // Initialize the variables to draw on the canvas
                this.mycanvas = document.querySelector("#mycanvas");
                this.cx = this.mycanvas.getContext("2d");

                // Task parameters
                this.nBlocks = 3;
                this.nTrials = 5;
                this.currentBlock = 0;
                this.currentTrial = 0;
```

```
		// Response handler flags
		this.waiting_for_task_start = 0;
		this.waiting_for_block_start = 0;
		this.waiting_for_trial_response = 0;

		// Timing variables
		this.currentTimeout = 0;
		this.time_stimulus = 0;
		this.time_response = 0;

		// Performance variables
		this.RT = 0;
		this.respKeys = ["F", "J"];
		this.respCodes = [];
		for (var i = 0; i < this.respKeys.length; i++) {
			this.respCodes.push(this.respKeys[i].charCodeAt(0));
		}
		this.pressedRespKeyIndex = -1;
		this.pressedRespKey = "";
		this.acc = 0;

		// Stimuli
		this.selectedStimCat = 0;
		this.selectedStimulusInStimCat = 0;
		this.nStimCat = 2;
		this.nStim = [4, 4];
		this.stimFilenames = [];
		this.stimuli = [];
		this.nStimLoaded = 0;

		// Task starts after stimuli are loaded
		this.init_stimuli();
	};

	// Stimulus loading

	this.init_stimuli = function() {
		// Just get the filenames
		for (var stimCat = 0; stimCat < this.nStimCat; stimCat++) {
			var subarr = [];
			for (var i = 0; i < this.nStim[stimCat]; i++) {
				subarr.push((function(i) {
					var fn0;
					if (stimCat == 0) fn0 = "Smilies/stim (" + (i + 1)
+ ").jpg";
					else fn0 = "Frownies/stim (" + (i + 1) + ").jpg";
					return fn0;
				})(i));
			}
			this.stimFilenames.push(subarr);
		}
		this.createNextStim(0, 0, []);
	}

	this.createNextStim = function(stimCat, stimIndexInCat, subarr) {
		if (stimIndexInCat < this.nStim[stimCat] && stimCat < this.nStimCat) {
			var img = document.createElement("img");
			img.src = this.stimFilenames[stimCat][stimIndexInCat];
			img.width = 200;
```

```
                img.height = 200;
                subarr.push(img);
                stimIndexInCat++;
                img.addEventListener("load", this.loaded_a_stim.bind(this));
                this.createNextStim(stimCat, stimIndexInCat, subarr);
        } else {
                this.stimuli.push(subarr);
                stimCat++;
                if (stimCat < this.nStimCat) {
                        subarr = [];
                        stimIndexInCat = 0;
                        this.createNextStim(stimCat, stimIndexInCat, subarr);
                }
        }
};

this.loaded_a_stim = function() {
        this.nStimLoaded++;
        var nStimTotal = 0;
        for (var n = 0; n < this.nStimCat; n++) {
                for (var m = 0; m < this.nStim[n]; m++) {
                        nStimTotal++;
                }
        }
        this.showText(this.nStimLoaded + " loaded of " + nStimTotal);
        if (this.nStimLoaded >= nStimTotal) {
                this.startTask();
        }
};

// Helper functions

this.showText = function(text0, x, y, color, fontsize) {
        if (color == undefined) {color = "white";}
        if (fontsize == undefined) {fontsize = 32;}
        this.cx.fillStyle = color;
        this.cx.font = fontsize + "px Georgia";
        this.cx.textAlign = "center";
        this.cx.textBaseline = "middle";
        let x_screen = (this.mycanvas.width / 2) * (1 + x);
        let y_screen = (this.mycanvas.height / 2) * (1 + y);
        this.cx.fillText(text0, x_screen, y_screen);
}

this.showPicture = function(img, x, y, picsize) {
        var w = this.mycanvas.width * picsize;
        var h = this.mycanvas.height * picsize;
        var l = (this.mycanvas.width / 2) * (1 + x) - w/2;
        var t = (this.mycanvas.height / 2) * (1 + y) - h/2;
        this.cx.drawImage(img, l, t, w, h);
}

this.emptyCanvas = function() {
        this.cx.fillStyle = "black";
        this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
};

// Task start

this.startTask = function() {
```

```
		this.showText("Smiley and Frownies", 0, 0);
		this.showText("Smiley: Press F", 0, 0.2);
		this.showText("Frownie: Press J", 0, 0.4);
		this.showText("Press a key to start", 0, 0.8);
		this.waiting_for_task_start = 1;
	}

	// Trials

	this.trial_init = function() {
		// Default values
		this.RT = 0;
		this.acc = 0;
		this.pressedResponseKeyIndex = -1;
		this.pressedRespKey = "None";
		// Randomize stimuli
		this.selectedStimCat = Math.floor(Math.random() * 2);
		this.selectedStimulusInStimCat = Math.floor(Math.random() *
this.nStim[this.selectedStimCat]);
		// Start first trial
		this.trial_fixation();
	}

	this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
	}

	this.trial_stimulus = function() {
		this.emptyCanvas();
		let imgToShow =
this.stimuli[this.selectedStimCat][this.selectedStimulusInStimCat];
		this.showPicture(imgToShow, 0, 0, 0.3);
		this.currentTimeout = setTimeout(this.trial_feedback.bind(this), 1000);
		this.waiting_for_trial_response = 1;
		this.time_stimulus = Date.now();
	}

	this.trial_feedback = function() {
		this.emptyCanvas();
		if (this.pressedRespKeyIndex == this.selectedStimCat) {
			this.acc = 1;
		} else {
			this.acc = 0;
		}
		if (this.acc == 1) {
			this.showText("Correct!", 0, 0, "green");
		} else {
			this.showText("Incorrect!", 0, 0, "red");
		}
		setTimeout(this.trial_ITI.bind(this), 500);
	}

	this.trial_ITI = function() {
		setTimeout(this.trial_end.bind(this), 250);
	}

	this.trial_end = function() {
		this.currentTrial++;
```

```
		if (this.currentTrial >= this.nTrials) {
			this.block_end();
		} else {
			this.trial_init();
		}
	}

	// Blocks
	this.block_init = function() {
		this.currentTrial = 0;

		this.emptyCanvas();
		this.showText("Block " + (this.currentBlock + 1), 0, 0);
		this.showText("Press a key to start", 0, 0.5);

		this.waiting_for_block_start = 1;
	}

	this.block_end = function() {
		this.currentBlock++;
		if (this.currentBlock >= this.nBlocks) {
			this.task_end();
		} else {
			this.block_init();
		}
	}

	// End of task
	this.task_end = function() {
		this.showText("End", 0, 0);
	}

	// Response handling
	this.respHandler = function(event) {
		this.time_response = Date.now();
		event.preventDefault();
		if (this.waiting_for_task_start == 1) {
			this.waiting_for_task_start = 0;
			this.block_init();
		} else if (this.waiting_for_block_start == 1) {
			this.waiting_for_block_start = 0;
			this.trial_init();
		} else if (this.waiting_for_trial_response == 1) {
			this.waiting_for_trial_response = 0;
			// Get RT
			this.RT = this.time_response - this.time_stimulus;
			// Get pressed response key, as an index in the respKey array
			this.pressedRespKey = "Illegal key";
			for (var i = 0; i < this.respCodes.length; i++) {
				if (event.keyCode == this.respCodes[i]) {
					this.pressedRespKeyIndex = i;
					this.pressedRespKey =
this.respKeys[this.pressedRespKeyIndex];
				}
			}
			clearTimeout(this.currentTimeout);
			this.trial_feedback();
		}
	};
```

```
		window.addEventListener("keydown", this.respHandler.bind(this));

}
```

This is pretty much all you need in terms of technical basics to make a whole lot of tasks, together with general JavaScript to tweak things like avoiding stimulus repetitions.

# Step 6: User IDs and saving data

We of course also want to have the data we're only ephemerally collecting now. To do so, we need to provide a User ID and we need to save the data. We're going to have to break out of JavaScript and into PHP for this.

**This means from now on we have to have PHP locally installed and active, e.g., via xampp, or work via online hosting. You can't continue from here without getting a PHP environment running one way or another.** If you're working on a local machine, you'll also need to open your file as a URL under PHP's localhost directory from now on.

The User ID will likely come from elsewhere, like Sona system or Qualtrics, and you'll need to capture it and feed it to the task. Usually this will involve a GET variable - a variable passed on in the URL itself, in text after a question mark. This looks like ?UserID=666 added after the normal URL.

For this step, we need to change task.html to task.php. Now open it in the browser and manually add a GET variable to the URL (so in the address bar of the browser type, e.g., task.php?UserID=666) and work with this GET-appended address from now on. Now adjust task.php as follows:

```
<html>
<head>
<?php
		$UserID = $_GET['UserID'];
		echo '<script>';
		echo 'var UserID = "'.$UserID.'";';
		echo '</script>';
?>

<html>
<head>

		<title>My Task</title>
		<script src = "task_code.js"></script>
</head>

<body onload="init()">
		<canvas id = "mycanvas" style = "background: black; display:block; margin: 0px auto;"
width = "600" height = "600"></canvas>

		<script>
			function init() {
				myTask = new Task();
```

```
                        myTask.init(UserID);
                };
        </script>
</body>

</html>
```

So we've added some PHP code to capture the GET variable UserID, put it into a PHP variable called UserID, and finally put it in a JavaScript variable also called UserID. We then feed that variable to the task init() function. We adjust that in the task code as follows to have the task know the user ID, which we'll use when saving data:

```
        this.init = function(UserID) {

                this.UserID = UserID;
                ...
```

To save data we need to create a separate PHP file: for security reasons JavaScript isn't allowed to change files on your computer (or host). We'll then call the PHP file to save data from in the Task; this uses something called AJAX, but it's fine as a black box for our purposes.

First we'll change the task to store and save a block's worth of data. This is going to into BlockData, an array with each element containing a key-value object with the data we want for each trial (e.g., stimulus category, RT, accuracy). This object is created and appended to the array in trial_end.

BlockData is cleared at the start of each block in block_init and we call the save function, saveBlockData, from block_end. This function is mostly technical boilerplate that turns the BlockData into a JSON string and sends the information to the PHP file for saving (if you have very long blocks this might need to be changed). The file is going to be named "MyTask_" + UserID + ".txt". You'll want to change "MyTask" to something more unique!

One important thing to set up: savedata.php expects a "results" subdirectory to save to, so make that now.

```
"use strict";

var Task = function() {

        this.init = function(UserID) {

                this.UserID = UserID;

                // Initialize the variables to draw on the canvas
                this.mycanvas = document.querySelector("#mycanvas");
                this.cx = this.mycanvas.getContext("2d");

                // Task parameters
                this.nBlocks = 3;
                this.nTrials = 5;
                this.currentBlock = 0;
                this.currentTrial = 0;
```

```
		// Response handler flags
		this.waiting_for_task_start = 0;
		this.waiting_for_block_start = 0;
		this.waiting_for_trial_response = 0;

		// Timing variables
		this.currentTimeout = 0;
		this.time_stimulus = 0;
		this.time_response = 0;

		// Performance variables
		this.RT = 0;
		this.respKeys = ["F", "J"];
		this.respCodes = [];
		for (var i = 0; i < this.respKeys.length; i++) {
			this.respCodes.push(this.respKeys[i].charCodeAt(0));
		}
		this.pressedRespKeyIndex = -1;
		this.pressedRespKey = "";
		this.acc = 0;

		// Stimuli
		this.selectedStimCat = 0;
		this.selectedStimulusInStimCat = 0;
		this.nStimCat = 2;
		this.nStim = [4, 4];
		this.stimFilenames = [];
		this.stimuli = [];
		this.nStimLoaded = 0;

		// Saving data
		this.BlockData = [];

		// Task starts after stimuli are loaded
		this.init_stimuli();
	};

	// Stimulus loading

	this.init_stimuli = function() {
		// Just get the filenames
		for (var stimCat = 0; stimCat < this.nStimCat; stimCat++) {
			var subarr = [];
			for (var i = 0; i < this.nStim[stimCat]; i++) {
				subarr.push((function(i) {
					var fn0;
					if (stimCat == 0) fn0 = "Smilies/stim (" + (i + 1) +
").jpg";
					else fn0 = "Frownies/stim (" + (i + 1) + ").jpg";
					return fn0;
				})(i));
			}
			this.stimFilenames.push(subarr);
		}
		this.createNextStim(0, 0, []);
	}

	this.createNextStim = function(stimCat, stimIndexInCat, subarr) {
		if (stimIndexInCat < this.nStim[stimCat] && stimCat < this.nStimCat) {
```

```
                        var img = document.createElement("img");
                        img.src = this.stimFilenames[stimCat][stimIndexInCat];
                        img.width = 200;
                        img.height = 200;
                        subarr.push(img);
                        stimIndexInCat++;
                        img.addEventListener("load", this.loaded_a_stim.bind(this));
                        this.createNextStim(stimCat, stimIndexInCat, subarr);
                } else {
                        this.stimuli.push(subarr);
                        stimCat++;
                        if (stimCat < this.nStimCat) {
                                subarr = [];
                                stimIndexInCat = 0;
                                this.createNextStim(stimCat, stimIndexInCat, subarr);
                        }
                }
};

this.loaded_a_stim = function() {
        this.nStimLoaded++;
        var nStimTotal = 0;
        for (var n = 0; n < this.nStimCat; n++) {
                for (var m = 0; m < this.nStim[n]; m++) {
                        nStimTotal++;
                }
        }
        this.showText(this.nStimLoaded + " loaded of " + nStimTotal);
        if (this.nStimLoaded >= nStimTotal) {
                this.startTask();
        }
};

// Helper functions

this.showText = function(text0, x, y, color, fontsize) {
        if (color == undefined) {color = "white";}
        if (fontsize == undefined) {fontsize = 32;}
        this.cx.fillStyle = color;
        this.cx.font = fontsize + "px Georgia";
        this.cx.textAlign = "center";
        this.cx.textBaseline = "middle";
        let x_screen = (this.mycanvas.width / 2) * (1 + x);
        let y_screen = (this.mycanvas.height / 2) * (1 + y);
        this.cx.fillText(text0, x_screen, y_screen);
}

this.showPicture = function(img, x, y, picsize) {
        var w = this.mycanvas.width * picsize;
        var h = this.mycanvas.height * picsize;
        var l = (this.mycanvas.width / 2) * (1 + x) - w/2;
        var t = (this.mycanvas.height / 2) * (1 + y) - h/2;
        this.cx.drawImage(img, l, t, w, h);
}

this.emptyCanvas = function() {
        this.cx.fillStyle = "black";
        this.cx.fillRect(0, 0, this.mycanvas.width, this.mycanvas.height);
};
```

```
	// Task start

	this.startTask = function() {
		this.showText("Smiley and Frownies", 0, 0);
		this.showText("Smiley: Press F", 0, 0.2);
		this.showText("Frownie: Press J", 0, 0.4);
		this.showText("Press a key to start", 0, 0.8);
		this.waiting_for_task_start = 1;
	}

	// Trials

	this.trial_init = function() {
		// Default values
		this.RT = 0;
		this.acc = 0;
		this.pressedResponseKeyIndex = -1;
		this.pressedRespKey = "None";
		// Randomize stimuli
		this.selectedStimCat = Math.floor(Math.random() * 2);
		this.selectedStimulusInStimCat = Math.floor(Math.random() *
this.nStim[this.selectedStimCat]);
		// Start first trial
		this.trial_fixation();
	}

	this.trial_fixation = function() {
		this.emptyCanvas();
		this.showText("+", 0, 0);
		setTimeout(this.trial_stimulus.bind(this), 250);
	}

	this.trial_stimulus = function() {
		this.emptyCanvas();
		let imgToShow =
this.stimuli[this.selectedStimCat][this.selectedStimulusInStimCat];
		this.showPicture(imgToShow, 0, 0, 0.3);
		this.currentTimeout = setTimeout(this.trial_feedback.bind(this), 1000);
		this.waiting_for_trial_response = 1;
		this.time_stimulus = Date.now();
	}

	this.trial_feedback = function() {
		this.emptyCanvas();
		if (this.pressedRespKeyIndex == this.selectedStimCat) {
			this.acc = 1;
		} else {
			this.acc = 0;
		}
		if (this.acc == 1) {
			this.showText("Correct!", 0, 0, "green");
		} else {
			this.showText("Incorrect!", 0, 0, "red");
		}
		setTimeout(this.trial_ITI.bind(this), 500);
	}

	this.trial_ITI = function() {
		setTimeout(this.trial_end.bind(this), 250);
	}
```

```
	this.trial_end = function() {
		// Store trial data as a new sub-array in BlockData
		this.BlockData.push({currentBlock: this.currentBlock, currentTrial:
this.currentTrial, stimCat: this.selectedStimCat, stimIndex:this.selectedStimulusInStimCat,
respIndex: this.pressedRespKeyIndex, RT: this.RT, acc:this.acc});
			this.currentTrial++;
			if (this.currentTrial >= this.nTrials) {
				this.block_end();
			} else {
				this.trial_init();
			}
	}

	// Blocks
	this.block_init = function() {
			this.currentTrial = 0;

			this.BlockData.splice(0, this.BlockData.length)

			this.emptyCanvas();
			this.showText("Block " + (this.currentBlock + 1), 0, 0);
			this.showText("Press a key to start", 0, 0.5);

			this.waiting_for_block_start = 1;
	}

	this.block_end = function() {
			this.saveBlockData();
			this.currentBlock++;
			if (this.currentBlock >= this.nBlocks) {
				this.task_end();
			} else {
				this.block_init();
			}
	}

	// Save data
	this.saveBlockData = function() {
			this.hr = new XMLHttpRequest();
			let dataString = "";
			for (var iTrial = 0; iTrial < this.BlockData.length; iTrial++) {
				let trialData = this.BlockData[iTrial];
				dataString = dataString + JSON.stringify(trialData) + "\n";
			}
			let fileName = "MyTask_" + this.UserID + ".txt";
			var vars = "fileName="+fileName+"&dataString="+dataString;
			this.hr.open("POST", "savedata.php", true);
			this.hr.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
			this.hr.send(vars);
	}

	// End of task
	this.task_end = function() {
			this.showText("End", 0, 0);
	}

	// Response handling
	this.respHandler = function(event) {
```

```
                        this.time_response = Date.now();
                        event.preventDefault();
                        if (this.waiting_for_task_start == 1) {
                                this.waiting_for_task_start = 0;
                                this.block_init();
                        } else if (this.waiting_for_block_start == 1) {
                                this.waiting_for_block_start = 0;
                                this.trial_init();
                        } else if (this.waiting_for_trial_response == 1) {
                                this.waiting_for_trial_response = 0;
                                // Get RT
                                this.RT = this.time_response - this.time_stimulus;
                                // Get pressed response key, as an index in the respKey array
                                this.pressedRespKey = "Illegal key";
                                for (var i = 0; i < this.respCodes.length; i++) {
                                        if (event.keyCode == this.respCodes[i]) {
                                                this.pressedRespKeyIndex = i;
                                                this.pressedRespKey =
this.respKeys[this.pressedRespKeyIndex];
                                        }
                                }
                                clearTimeout(this.currentTimeout);
                                this.trial_feedback();
                        }
                };

                window.addEventListener("keydown", this.respHandler.bind(this));

}
```

Now we create the PHP file called savedata.php being loaded in the save function. Importantly, this saves to a simple ASCII text file, and not all hosting companies will let you use that function. Dreamhost so far does, for example. It would maybe be better to a SQL database but I like to work with my little text files.

```
<?php
        $fileName = "results/".$_POST['fileName'];
        $dataString = $_POST['dataString'];
        $fp = fopen($fileName, "a");
        fwrite($fp, $dataString."\n");
        fclose($fp);
?>
```

Try it out and check whether sequences of correct / incorrect and fast / slow are correctly saved. The format is JSON, which any mainstream programming language will be able to work with.

This is a very basic file and you might want to embellish it with security checks. But: it's working! You can now send a participant an invitation with a link containing a GET variable with their UserID; they can do the task online (if it's hosted somewhere); and you'll have the data afterwards.

# Step 7: Emails

It can be convenient to know when someone has completed the task; we also want the participant to be clearly instructed they've finished the session. To do both things, when the task ends we'll make the task go to a new PHP webpage which will send us an email and display a goodbye message to the participant. This emailing method is quite unreliable though, and messages could well be sent to spam, or even invisibly blocked as untrusted email by email services. A generally better solution, beyond the scope here, is to install and use Swiftmailer.

That said, all you have to do to try the simple method is add the following line to the task_end function (I've also added an emptyCanvas() call that was previously missing):

```
// End of task
this.task_end = function() {
	this.emptyCanvas();
	this.showText("End", 0, 0);
	window.location = "end.php?UserID=" + this.UserID;
}
```

You might also want to set nBlocks = 1 and nTrials = 3 to skip through quickly. Now at the end of the task, the webpage end.php will open, with the UserID added as a GET variable. The page end.php looks like this, but you'll have to fill in your own email-address and website:

```
<?php
	$UserID = $_GET['UserID'];
	$subject = $UserID." has completed MyTask\n";
	$to = "me@email.com";
	$from = "me@email.com";
	$headers = "From: mywebsite.com\r\nReply-To: ".$from;
	$mail_sent = @mail($to, $subject, $message, $headers);
?>

<html>
<head>
	<title>My Task: Complete!</title>
</head>

<body>
<h1>Thank you!</h1>
You have completed the experiment and can close the tab or browser.

</body>

</html>
```

Finally, if you want to send automated emails, very briefly: use a cron job that calls a PHP script that uses files you create at the end of each session, for example. In Dreamhost, you just specify in the control panel where your cron file, cron.sh, is, and how often you want to call it. It would contain lines of code like this:

```
#!/bin/sh
curl "https://www.tegladwin.com/Test/2018_12_04_AMAT_S/Online/mailsender.php"
```

# Next Steps

The example task used here is a dot-probe task. One thing to be aware of is that the bias scores you tend to get from this task have very low reliability. In general, especially if you want to use performance scores as an individual difference variable rather than just testing within-subject effects, reliability is a potential issue with any implicit measure. Online data collection can make this issue worse, or at least make reviewers think so. Some researchers argue that this means we need to move to eye tracking, which in itself is a great approach, but obviously this would have serious practical implications. A general shift away from behavioural measures of biases would be premature, as it turns out reliability can be fine if you tweak the task. This *exploration* of a next generation of implicit measures – following the maybe premature *exploitation* of traditional tasks – is one next step to take once you can program these tasks. For more information, see the section "Anticipatory automaticity and a reliable spatial attentional bias task" on tegladwin.com and:

> Gladwin TE, Vink M (2020). Spatial anticipatory attentional bias for threat: Reliable individual differences with RT-based online measurement. Consciousness and Cognition, 81, 102930, doi:10.1016/j.concog.2020.102930. Preprint at https://psyarxiv.com/agqp6/

Of course you have to consider data quality and ethics when collecting data online from, e.g., MTurk, but it doesn't seem that these factors are *especially* problematic, see, e.g.,:

> Hauser Common Concerns with MTurk as a Participant Pool: Evidence and Solutions, https://psyarxiv.com/uq45c/.
>
> Chetverikov, A., & Upravitelev, P. (2016). Online versus offline: The Web as a medium for response time data collection. Behavior Research Methods, 48(3), 1086–99. https://doi.org/10.3758/s13428-015-0632-x
>
> van Ballegooijen, W., Riper, H., Cuijpers, P., van Oppen, P. , & Smit, J. H. (2016). Validation of online psychometric instruments for common mental health disorders: a systematic review. BMC Psychiatry, 16(1), 45. https:// doi.org/10.1186/s12888-016-0735-7

Another next step is to adjust tasks like this to be training tasks, aimed at automatization responses or changing response tendencies to certain types of stimuli. If you merely adjust the probability of whether the probe stimulus appears at the location of the angry faces, you have what's called an Attentional Bias Modification training. Just like with implicit measures, there's a lot of work to be done exploring which variants of this might work best, or at all. For instance, the anticipatory version of the dot-probe task has a training variant that showed hypothesized effects in a first study:

Gladwin TE, Möbius M, Becker ES (2019). Predictive Attentional Bias Modification induces stimulus-evoked attentional bias for threat. Europe's Journal of Psychology, 15(3), 479-490, https://doi.org/10.5964/ejop.v15i3.1633.

One final thing to think critically about is timing. Running the bits of code needed for each event isn't instantaneous, so this can affect timing; responses aren't registered instantaneously; stimuli only start to be sent to the actual screen after the function ends. Whether this matters for your particular study or is basically a drop in the ocean of noise due to a human doing the task is a case-by-case judgment; obviously hardcore psychophysics studies you'd usually do in a darkened lab with specialized equipment might not be best done in JavaScript and online. However, at least some effects seem reliable and robust.

And that's it for this tutorial! See tegladwin.com for updates, corrections or contact, and for the completed files for each step.