# Welcome to the Study on Evaluating Explicit Programming Strategies

In the following slides, you'll learn some key concepts that will help you in completing the study, work on a few hands-on exercises to try out your understanding of the concepts, and then begin the two study tasks. As you go through the slides, you should make sure you understand the concepts and ask the experimenter if you are confused.

# Background survey

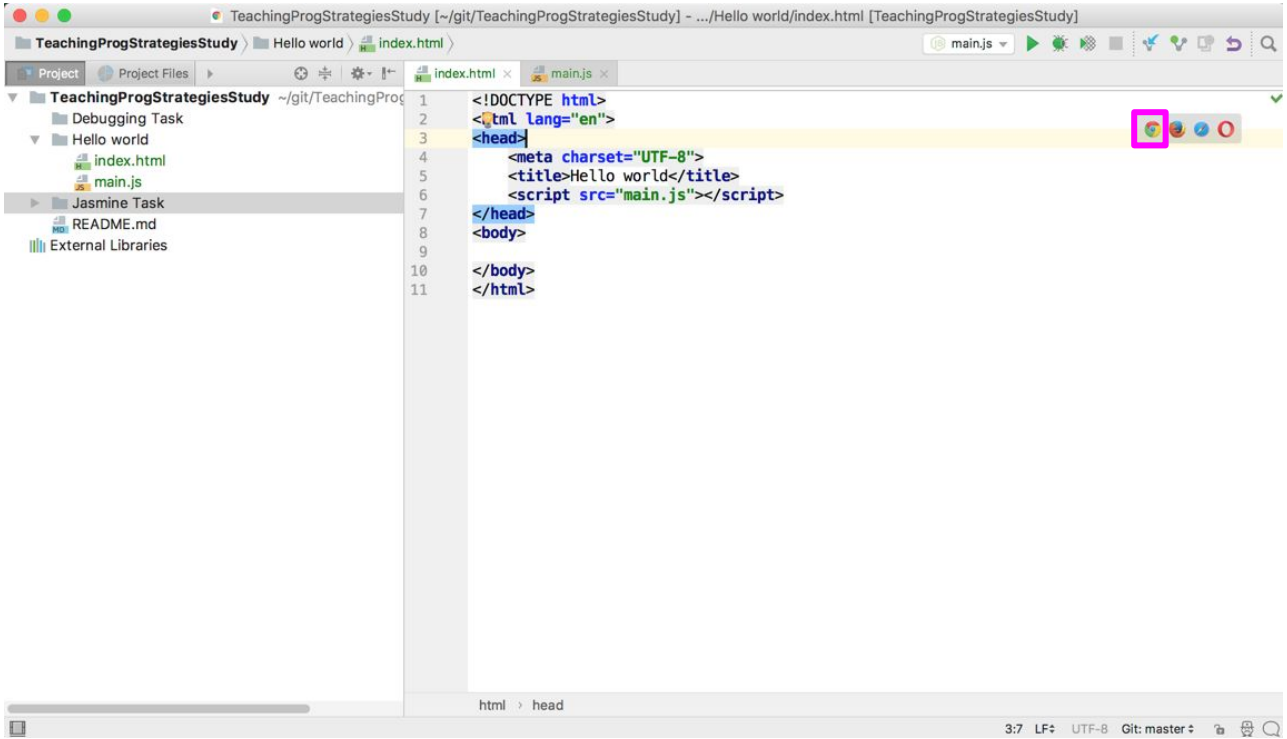
Please complete the following survey:

https://goo.gl/forms/1hfffUdTAf0xDo5f1

# WebStorm

WebStorm is an IDE for editing, testing, and debugging code.

To run a project, you should (1) navigate to the **HTML** file in the current directory and (2) click the Chrome browser **button** in the right hand edge of the code window.

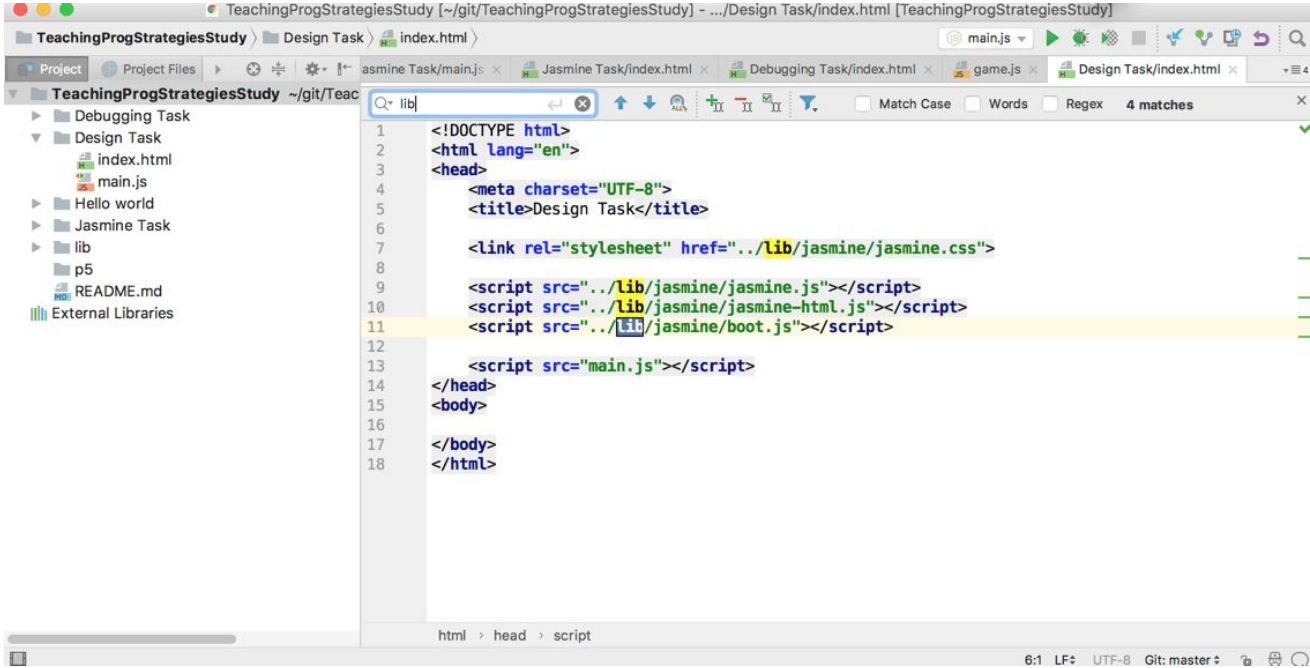


# Find in file

Press COMMAND F to search within a file.

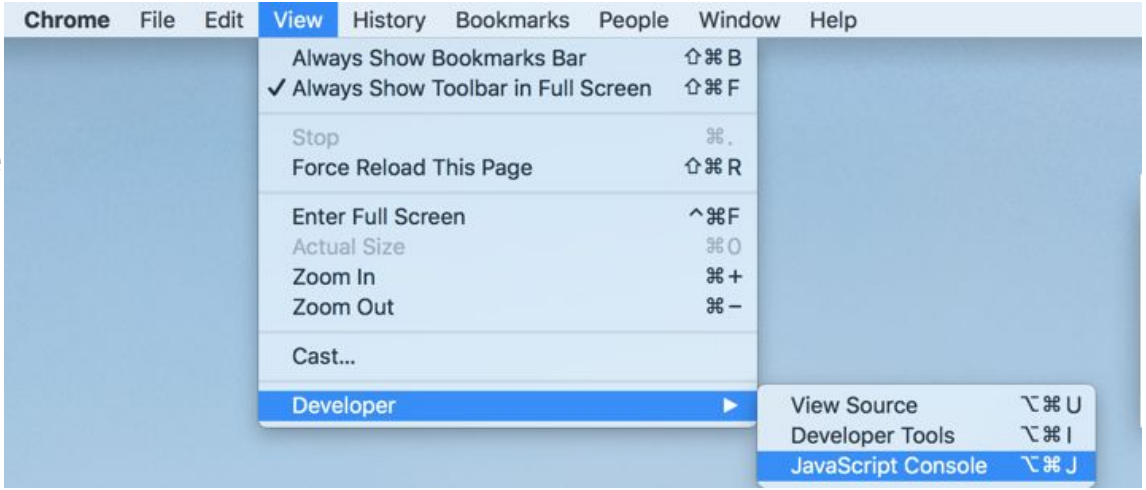
Each match is highlighted with a yellow background.

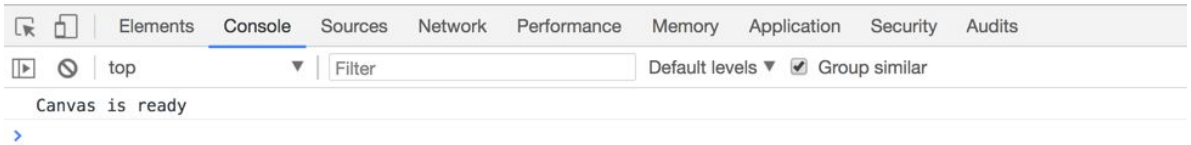
Press ENTER to move between matches.



# Open the Console in Chrome

To see output to the Console in Chrome, go to View / Developer / JavaScript Console.



# Hello world

Write a program that prints ‘Hello world’ to the console. Verify that the program works correctly by opening the console.

# Unit testing with Jasmine

In this study, you will use the Jasmine unit testing framework to write unit tests.

Here's a simple example of a Jasmine unit test:

```
describe("A suite is just a function", function() {
  var a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toEqual(true);
  });
});
```

Declare a set of Jasmine tests

Declare a specific test

Test that the variable a is equal to true.

# Strategies

A strategy is a high-level plan for accomplishing a task, describing a series of steps to take to accomplish a goal.

Strategies are a common part of everyday life, describing steps with which to find an email in an email client, cook the perfect pizza, or tie your shoes.

Programming strategies describe how software developers do similar everyday tasks, like different ways to debug a defect or go about software design.

# Jasmine Exercise

Write a test for the following add function

```
function add(a, b)
{

    return a + b;

}
```

# Learning expert strategies

Expert developers know specific strategies that let them perform challenging and complex programming tasks more quickly and easily.

In this study, you will learn two of the programming strategies used by experts by using a tool that instructs you step by step in using programming strategies: StrategyTracker.

# Working with strategies in StrategyTracker

StrategyTracker is a web application that helps you work together with a computer to learn a strategy.

Like in a programming language, strategies are described through a sequence of steps that you can use to accomplish a task. For example, an Execute statement describes an action for you to take in the world.

*Example*

Clap your hands.

# Variables

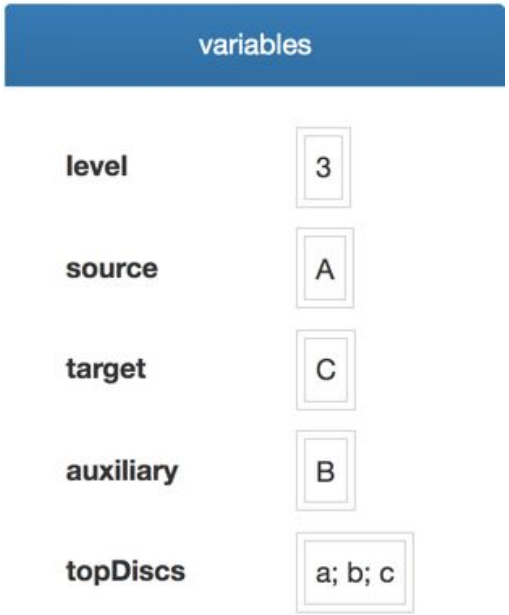
StrategyTracker lets you write down variables, letting you keep track of information as you work. Like a variable in a programming language, a variable can be assigned a value, and you can read the value of a variable. But in StrategyTracker, variables are assigned a value by you, the user, and have their values read by you. The computer helps by keeping track of the value that you assign a variable so it's there when you need it next. Variables are identified in strategies by a single quote around the name: 'aVariableName'.

*Example*

set 'weather' to the current weather outside

# Writing down variables

StrategyTracker lets you record the value of a variable in the variables pane. You can then see the value of the variable that you've recorded. Variable values are text and can be as long (or as short) as you'd like.



# Conditionals

In some situations, a strategy may have different steps to perform, depending on some condition that is true in the world. To handle such situations, StrategyTracker includes an IF statement. Just like in a programming language, an if statement examines a condition to determine which statement to execute next. Unlike in a traditional language, in StrategyTracker, **you** make the decision about if the condition is or is not true, which you then communicate to the computer by clicking the appropriate button. Conditionals may reference the value of variables you have set before.

*Example*

if ‘weather’ is currently nice 

# Loops

As in a traditional programming language, sometimes a strategy may require you to perform a set of steps multiple times. StrategyTracker includes two types of loops. FOR EACH loops loop over elements of a collection, repeating a set of steps for each element. UNTIL loops repeat a series of steps until an exit condition becomes true. FOR EACH loops are handled entirely by the computer, as it selects an element for each iteration. Like an IF statement, until loops will ask you to make a decision about if a condition is or is not true.

*Example*

until ‘weather’ is nice enough for a walk 


Read a book for 15 minutes

# Substrategies

Sometimes strategies contain many steps. Just as in traditional programs, long strategies may be broken up into separate substrategies, each reflecting part of the overall strategy. To connect each substrategy together, a DO statement describes the substrategy to be started next. As in a traditional programming language, the call itself is an instruction to start performing another strategy. In StrategyTracker, it will transfer control to the new strategy and list the set of steps for the new strategy so that you can perform the steps. **When you find a call to a substrategy, you should NOT start doing the steps until you step into the substrategy.**

*Example*

```
do takeAWalk('destination')
```

# Variables in Substrategies

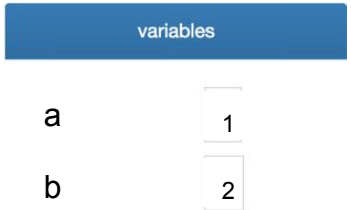
Just like in a debugger, when you step into a substrategy, the variables pane on the right changes to show the variables that are now in scope for the current strategy.

*Example*


add(1, 2)

Strategy add(a, b)

return a + b



# Return

Like in a traditional programming language, substrategies may have parameters and may return a value. In this case, there’s nothing that you need to do. StrategyTracker will automatically copy the values you have set for each of the variables. When a substrategy is finished, StrategyTracker will automatically copy the value returned back to the original strategy that invoked it.

*Example*

return ‘ideasYouHadOnWalk’

# Beginning a strategy

To begin a strategy, you will first be asked to set some values for variables. This information is needed to initialize the strategy with information on the task you are to perform. StrategyTracker will prompt you with some background on the information you should provide.

*Example*

# Where are you taking a walk to?

destination the park down the street

# Warning: Self-regulation

When doing a programming task, you may have a strategy that you've always used in the past. Perhaps, when faced with a null pointer exception, the first thing you **always** do is set a breakpoint and run the code in the debugger. To successfully use StrategyTracker to learn a strategy, you will need to stop whatever you would normally do and instead do the steps **exactly** as described in the strategy. In this way, you will be able to try out approaching the problem just as an expert developer might approach the same problem.

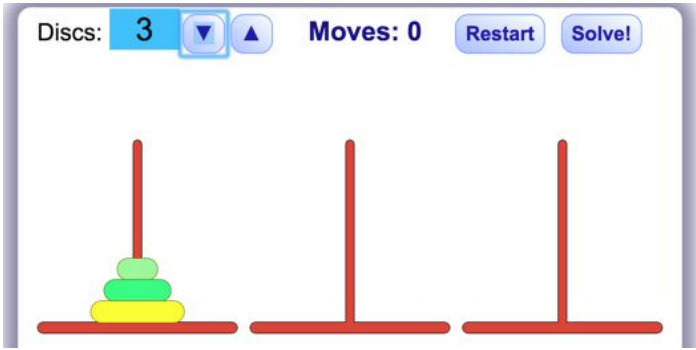
# Example: Git Merge

The experimenter will demo a strategy for merging in Git.

# Example: Tower of Hanoi

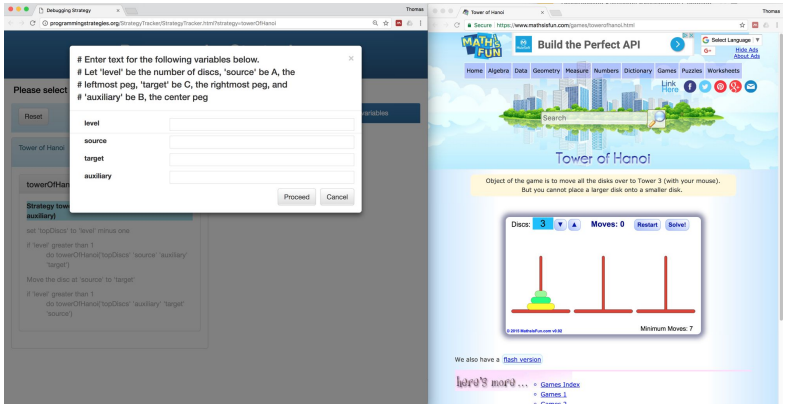
The tower of hanoi is a simple puzzle game. The objective is to move all of the discs while following three rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
3. No disk may be placed on top of a smaller disk.




How can you solve this puzzle? You'll follow a strategy in StrategyTracker to learn how.

# StrategyTracker Example: Tower of Hanoi




Open the following two links in separate windows and place them side by side like in the screenshot above. Follow the directions in StrategyTracker step by step to solve the puzzle. If you forget to follow the steps in StrategyTracker, the experimenter will help you by reminding you to follow the strategy step by step.

Click here to start the Tower of Hanoi strategy in StrategyTracker

Click here to start the Tower of Hanoi

# End of StrategyTracker tutorial

You've now completed all of the tutorials and learned how to use StrategyTracker. The most important thing to remember in using StrategyTrack is to practice **self-regulation**.

StrategyTracker helps you by teaching you a new strategy. This means that it's important to do the steps exactly as described in StrategyTracker. If you do not follow the steps, you won't be learning the new strategy StrategyTracker tutorial.

# Task 1: Learn a Debugging Strategy (30 minutes)

In task 1, you will learn a new debugging strategy.

To help you learn the strategy, you'll try it out with an actual defect.

# Task 1: Learn a Debugging Strategy (30 minutes)

To try out the strategy, you'll work through the strategy step by step to fix a problem with a simple web-based game.

The way the game is supposed to work is that the snake moves up, down, left, and right (using the keyboard). Every time the snake eats a dot, it grows in length by one. If the snake collides with itself, the game is over.

As you'll see when you play the game, the snake does not move up, down, left, and right. It just seems to move diagonally, and when you press the arrow keys in certain directions, the game ends.

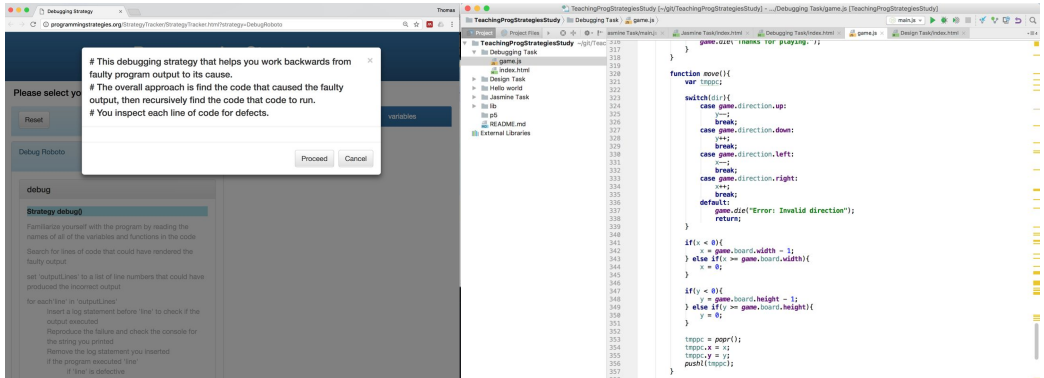
The experimenter will demo what the game should look like when it functions correctly.

# Task 1: Learn a Debugging Strategy (30 minutes)

Learn a new debugging strategy by trying it out to find and fix a failure. Use the StrategyTracker tool step by step to figure out what to do next at each point. You have up to 30 minutes. When you've followed the strategy to the end and believe you've found a fix, tell the experimenter which text on which line of code you changed (e.g., line 27, replace "hello" with "goodbye").

Open the Debugging Task directory in WebStorm, and open the following link and place it side by side with the WebStorm window.

[Open the debugging strategy to learn in a new tab]



# Task 1: Debriefing

1) Describe the strategy or strategies you used to make progress on the task.

2) How did this strategy help in making progress on the task?

3) In what ways, if any, did the strategy get in the way of making progress?

# Task 2: Learn a Design Strategy (30 minutes)

In task 2, you will learn a new design strategy.

To help you learn the strategy, you'll try it out with an actual design problem

# Task 2: Learn a Design Strategy (30 minutes)

To try out the strategy, you'll work through the strategy step by step to design and implement a simple autocomplete feature.

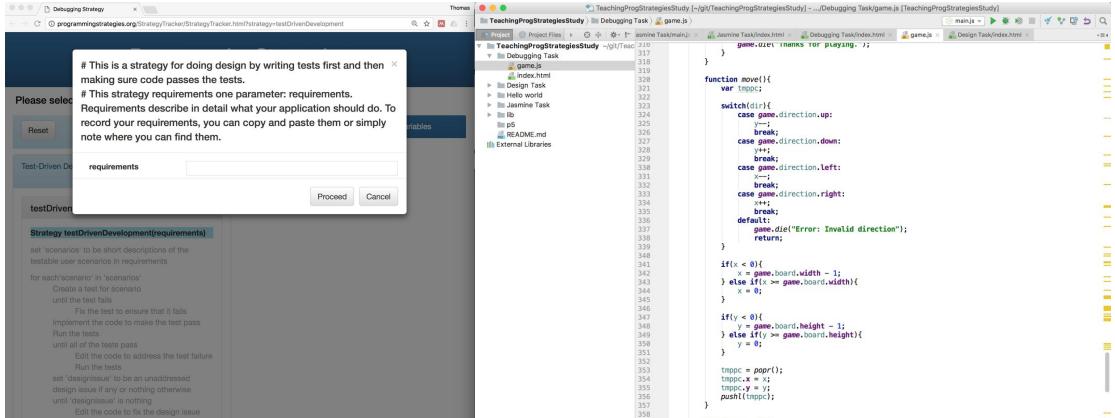
Whenever the user begins typing a new word, your autocomplete will recommend possible completions. Your autocomplete will generate completions based on the words that the user has already entered in the text area, ranking valid possible completions from most to least likely based on the frequency of the word in existing text. If there are no possible completions based on these words, your system should generate an empty list of completions.

# Task 2: Learn a Design Strategy (30 minutes)

Learn a new design strategy by trying it out to design and implement a simple autocomplete. Use the StrategyTracker tool step by step to figure out what to do next at each point.  Your goal is to build a working implementation and to craft a clear and easy to maintain design. You have up to 30 minutes. Notify the experimenter when you're done.

Open the Design Task directory in WebStorm, and the following link and place it side by side with the WebStorm window.

[Open the design strategy in a new tab]



# Task 2: Debriefing

1) Describe the strategy or strategies you used to make progress on the task.

2) How did this strategy help in making progress on the task?

3) In what ways, if any, did the strategy get in the way of making progress?

# Study Debriefing