Mystery Functions

by Amy Huang

Computer Science Honors Thesis

Brown University, May 2020

Advisor

Tim Nelson

Brown University Computer Science

Reader

Shriram Krishnamurthi

Brown University Computer Science

In collaboration with

Rob Goldstone

Indiana University Psychological and Brain Sciences

Abstract

This thesis project investigates the process of theory formation through a study design we call Mystery Functions. Previous works have investigated how people collect and respond to data that they retrieve from a central "oracle" entity. Under this design paradigm, participants are given the means to query an all-knowing source of truth for data about some observed phenomenon with the goal of coming up with a theory that totally explains it. The oracle answers all queries immediately and perfectly, giving exactly the data asked for. To determine whether they have succeeded, participants present their theory to the oracle, and the oracle tells them whether or not it's correct.

We designed an activity after this model in which participants guess what a computational function does. At first, the only pieces of information they know are the number of inputs and the input and output types of the function; then, they are given the means to ask for the corresponding output of any input and make a guess about the function. The activity is implemented as a web application. We conducted a study with 68 students from a psychology subject pool at Indiana University Bloomington, and 80 students from a software engineering class at Brown University.

Our contributions are the dataset of inputs evaluated, quiz attempts made, and labeled guesses about the functions, and our analysis. Though the quality of guesses given by the two groups of students varies substantially, both groups exhibit similar patterns, such as doing most of their data collection before making their first guess. While collecting data, they made repetitive changes to past queries for data to generate future queries.

Contents

May 28, 2020

1 Introduction

People form theories based on observations they make about their surroundings, and often seek to systematically collect data and refine those theories. How people go about designing and executing this data collection is of great interest in the fields of psychology and cognitive science [2, 7].

One way to study the process of theory formation is to create an all-knowing source of truth, or "oracle", that subjects query for information. The oracle knows everything about some phenomenon in the subjects' environment and will answer any question about it instantly and accurately. The subjects come up with a theory about what causes the phenomenon by making queries of their choice. At any time, subjects can present their theory to the oracle, and the oracle will respond whether it is correct. With this activity, we can observe how people design experiments and react to the results.

In our project, the phenomenon to be understood is a computational function that takes in data, performs computation, and returns the result. The subject's goal is to find out what the function is. They can only make queries of the following form: "What does this function output for an input of X?", where X is some well-formed input value. We call this activity Mystery Functions, and implement it using a web application.

Past works focused narrowly on how people understood mathematical functions that took in a single real number and output another [1, 5], giving subjects initial data to analyze but little ability to explore further [3, 6]. Subjects were repeatedly shown examples of input-output pairs for several similar functions, and then demonstrated their comprehension by guessing the outputs of a function that they'd never seen before, but was similar to those

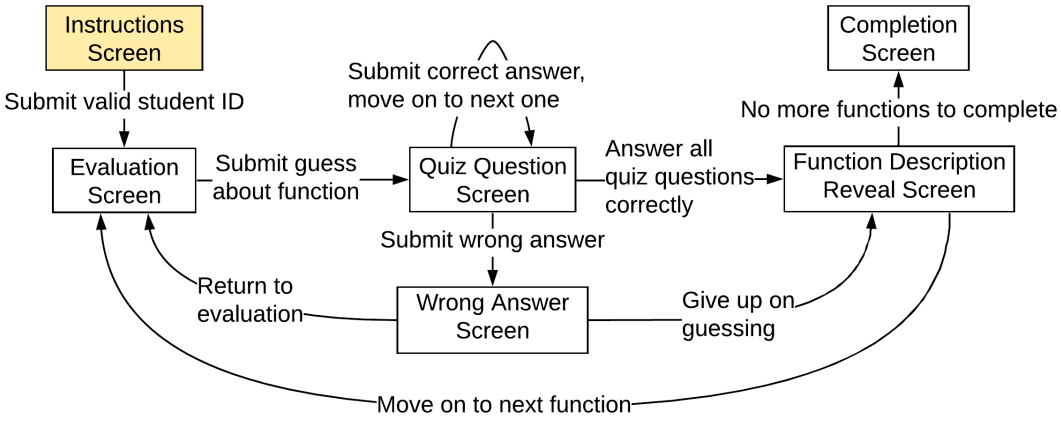
presented.

It is shown that having agency over the data collection process may help people learn better than passively being given the same data [4]. In our design, subjects can easily design and execute plans for data collection, and choose freely when to switch between data collection and venturing guesses about the phenomenon. Furthermore, we ask subjects to explicitly articulate their guess about the phenomenon in their own words, allowing insight into the conscious thought processes of theory formation.

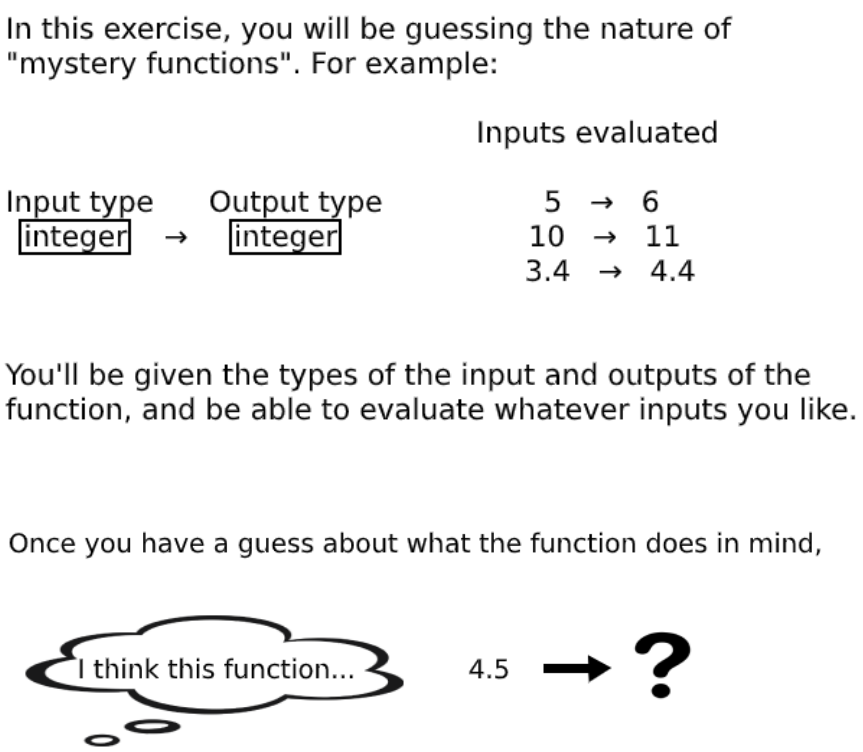
2 Study Design

We now describe the workflow of a mystery functions session. Stepping through the actions of a hypothetical subject named Hester, we describe the possible actions they can take at each stage of interaction with the web application. For the purpose of demonstrating the mystery functions workflow, the mystery function will be simple: given a list of integers, the function returns the absolute value of the sum of its elements. The functions actually used in the study are detailed later in Functions Given (Section 3).

Hester receives the URL for the study web application, and opens their web browser to it. They are taken to the Instructions Screen.

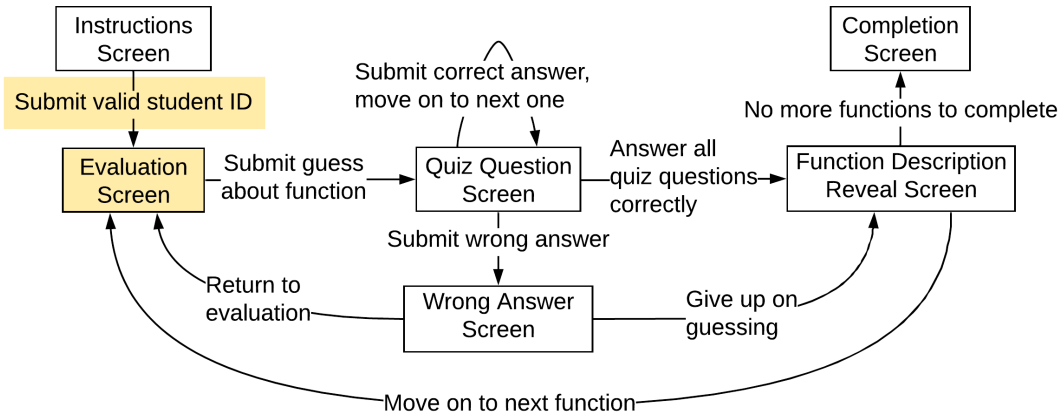


They read the below description of Mystery Functions.

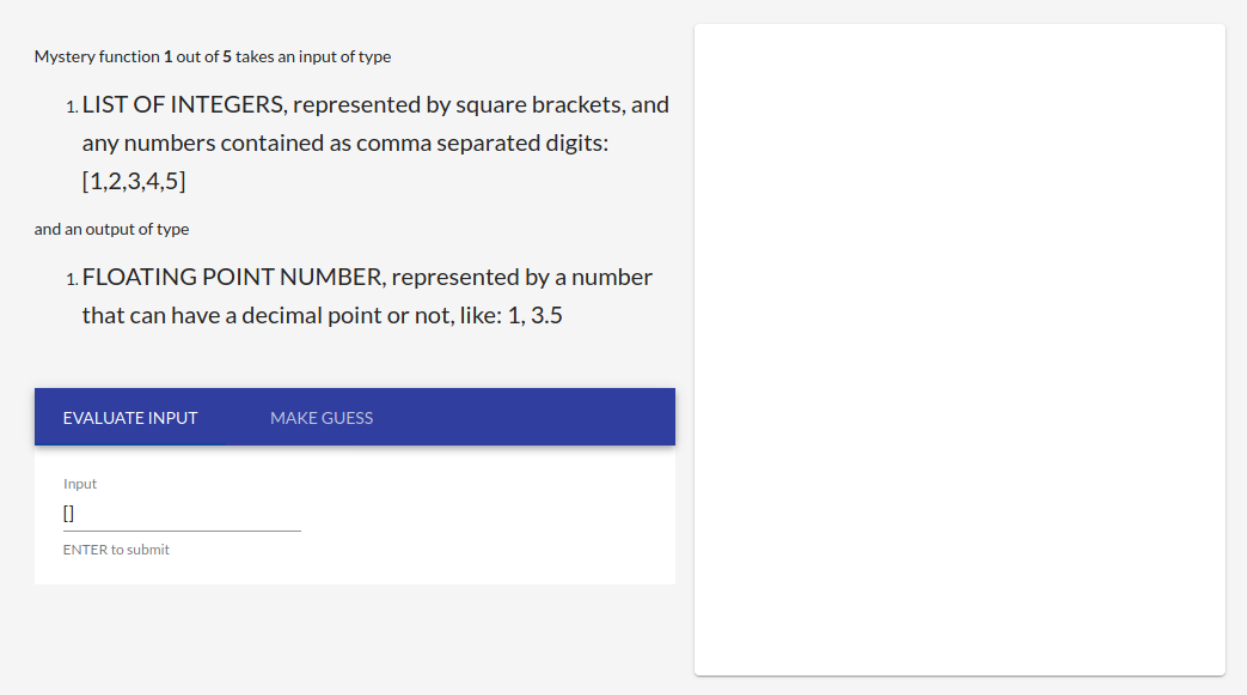


Hester is prompted to submit an ID that we will use to uniquely identify their session, so that we can give them academic credit for it. The subject populations we drew from are detailed further later in Subjects (Section 4).

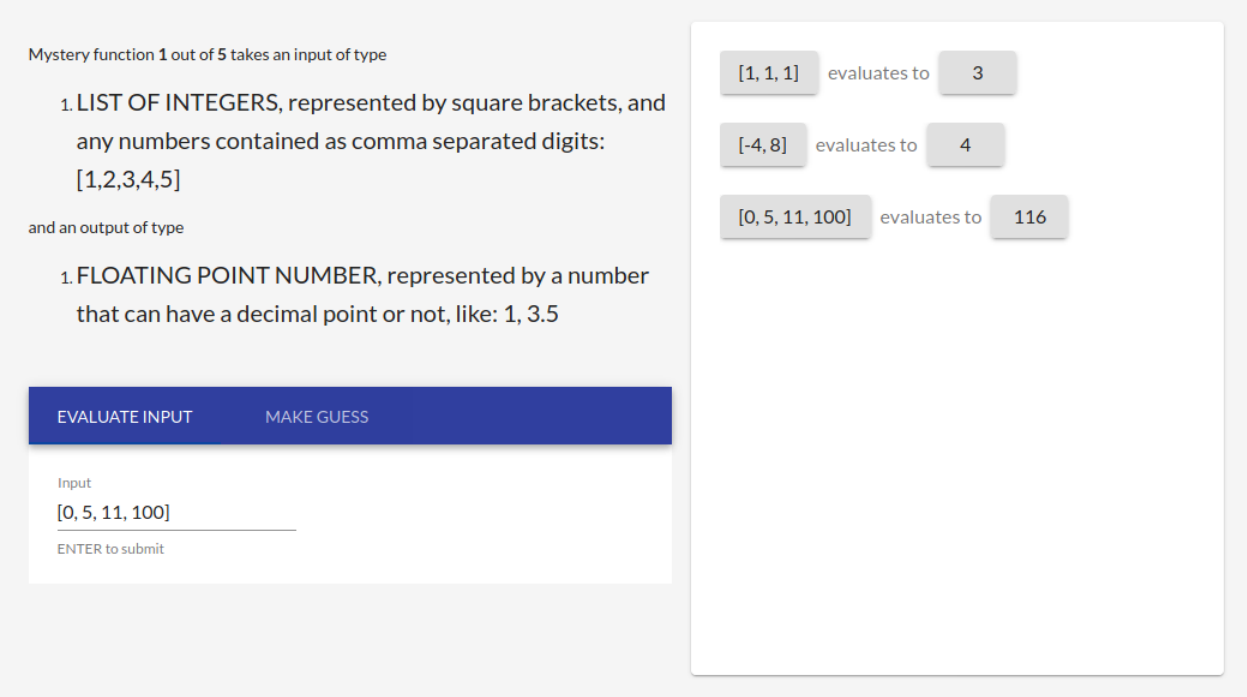
Upon entering an ID, Hester arrives at the Evaluation Screen, where they can evaluate inputs.



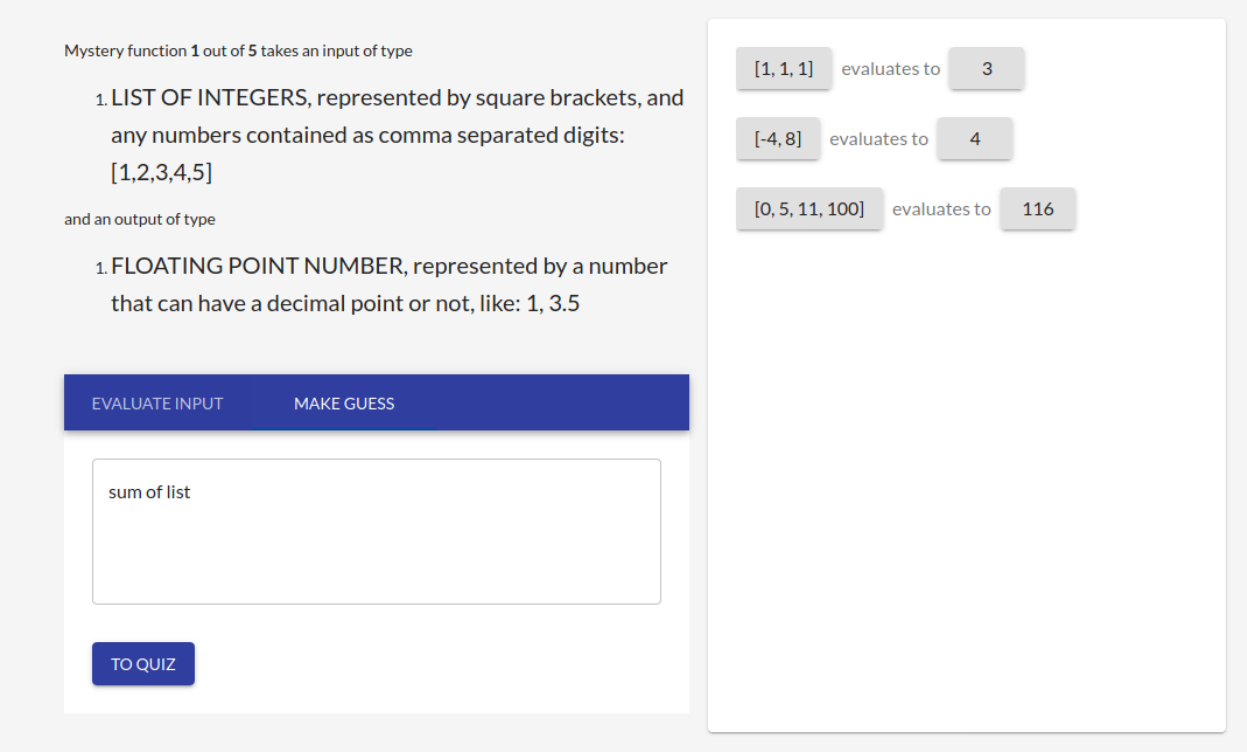
In the top left corner of the screen is a description of the function signature, and how to format values of the input type as text. For lists of integers, there must be square brackets indicating the beginning and end of the list and commas separating list elements. In the bottom left corner, there is a tabbed window. The first tab is for evaluating inputs, with a text box whose contents can be submitted by pressing Enter. The second tab is for submitting guesses about what the function does. On the right side of the screen, a console displays the input-output pairs of inputs evaluated. There is no limit to how many inputs can be evaluated, nor how much time Hester has to complete the function.



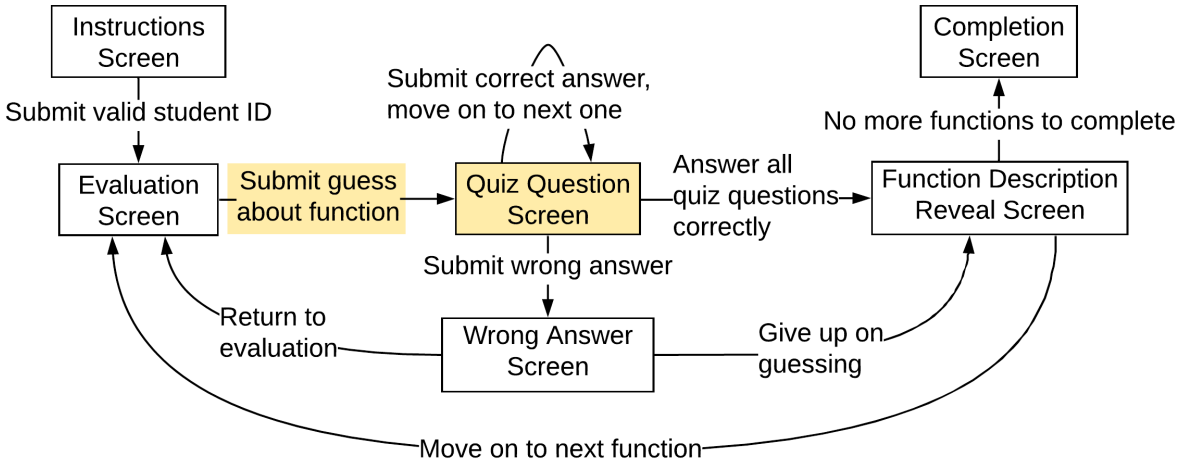
Hester evaluates inputs [1, 1, 1], [-4, 8], and [0, 5, 11, 100] which produce the outputs 3, 4, and 116, respectively. The application logs each input evaluation, sending the user ID, local browser time, the input, and the output to the web server.



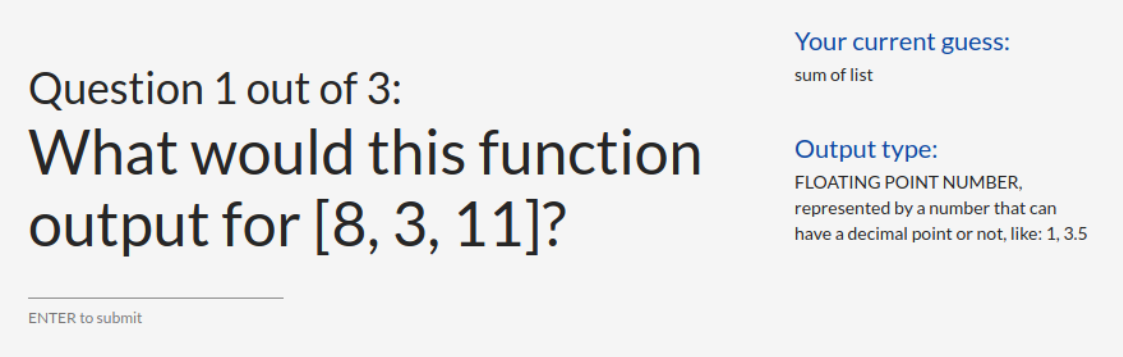
Hester thinks that the the function outputs the sum over the input list, and decides to submit a guess about the function. They navigate to the second tab on the window in the bottom left corner, and write out "sum of list".



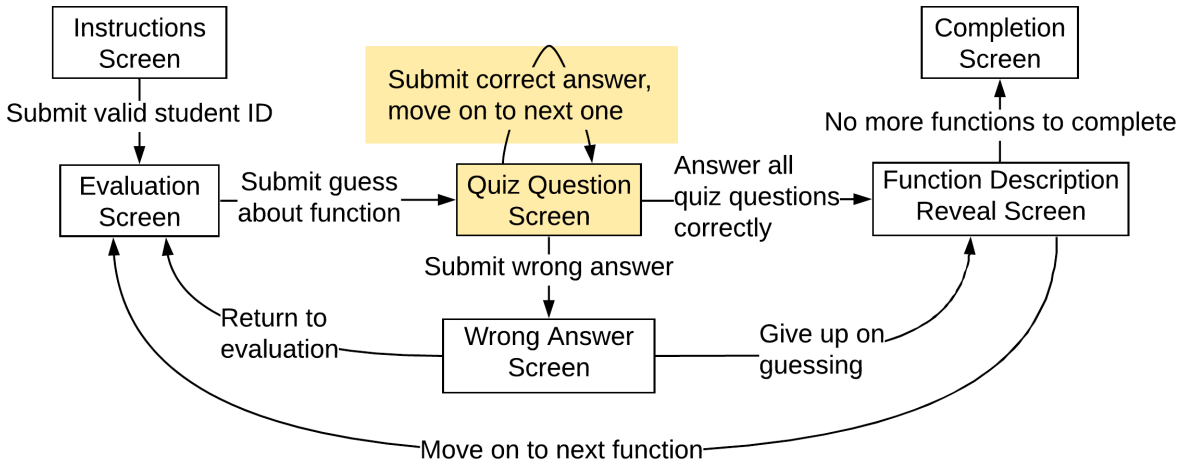
Upon submitting, they are taken to the Quiz Screen.



3 questions are presented, one at a time, each of the same form: "Given this input, what value would the mystery function return?". For every quiz attempt, the quiz question inputs remain the same and are presented in the same order.

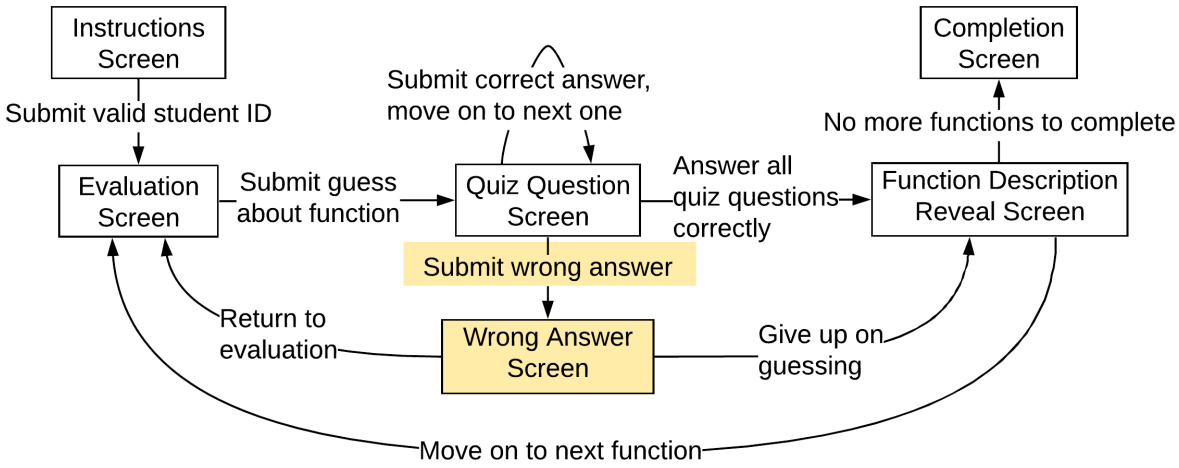


The first input presented is [8, 3, 11]. The correct answer (the absolute value of the sum of the list) is 22. Hester gets this question right.

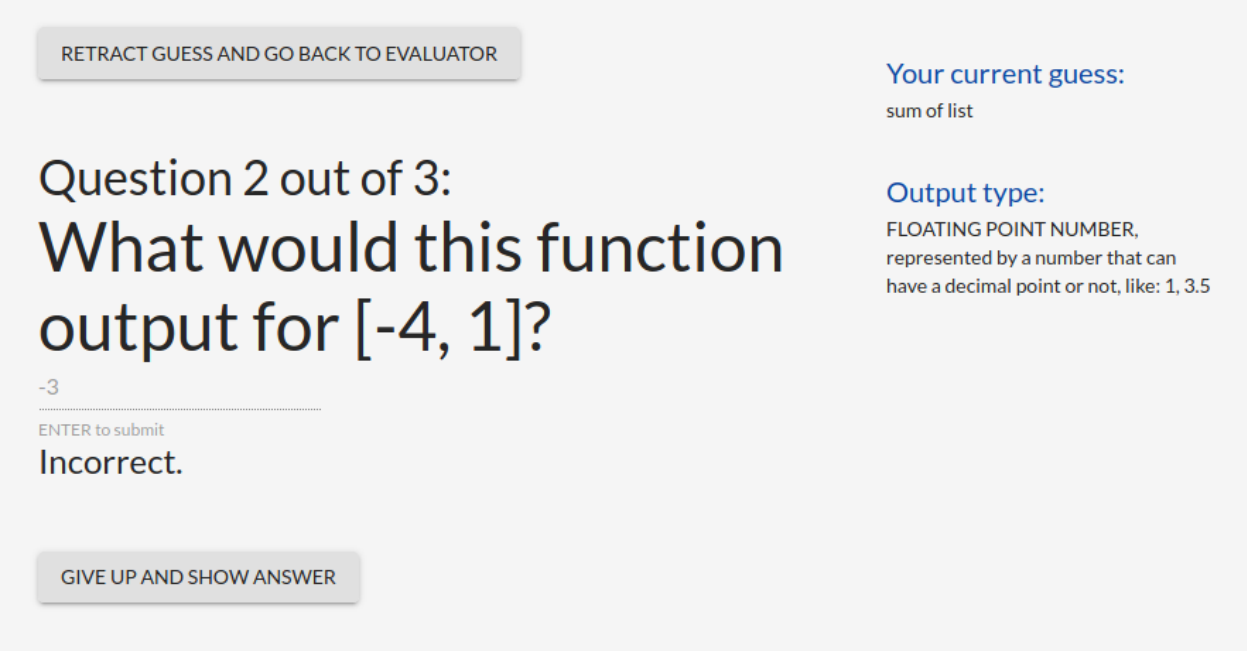


The application logs the action, saving the ID and time again, as well as the input presented, Hester's submitted output, and the actual output corresponding to the input. They are prompted to move to the next quiz question.

The next input presented is [-4, 1]. The correct answer is 3, but Hester answers -3.

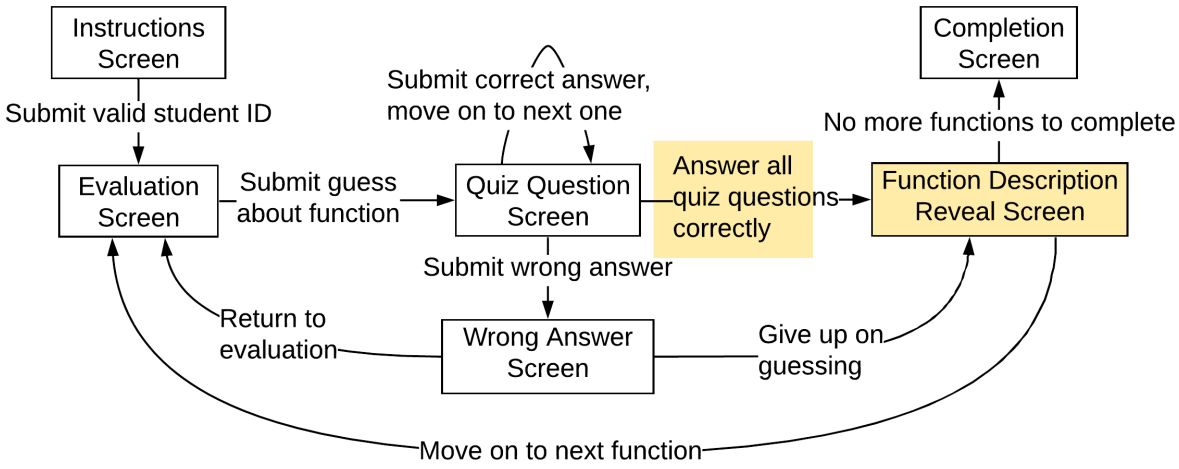


The application shows two options now: Hester can either return to the Evaluation Screen and resume evaluating inputs, or give up on guessing the current mystery function and move on to the next one.

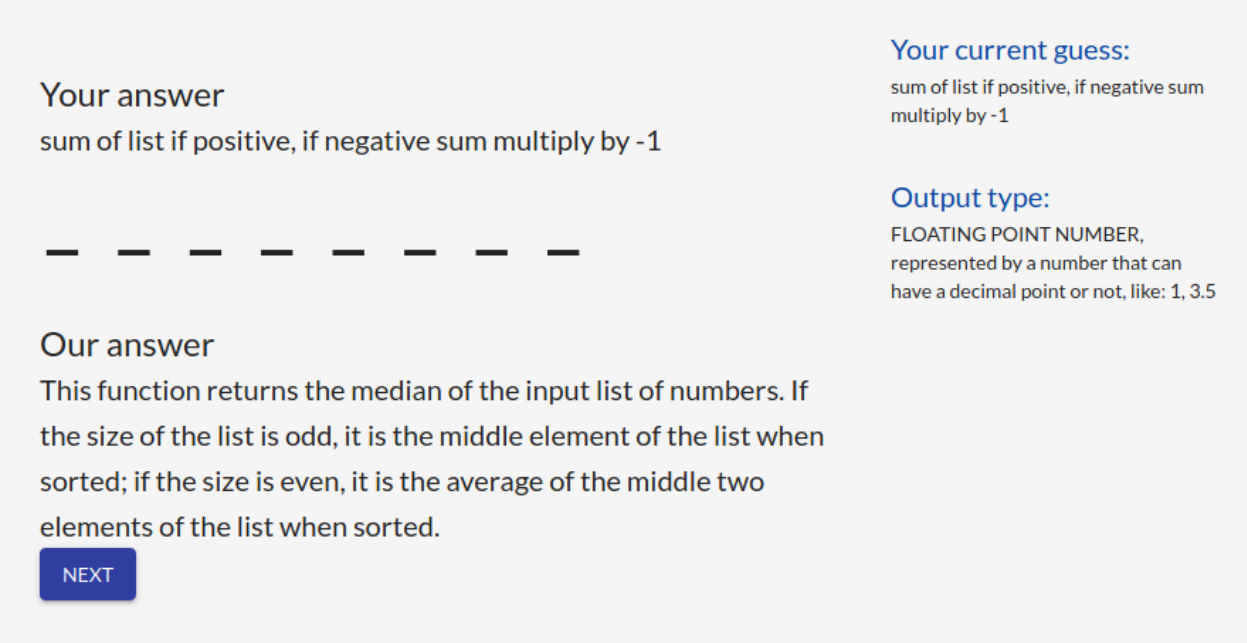


Hester chooses to go back to the Evaluation Screen, where the previous inputs evaluated are still displayed. They evaluate [-3, -2] and [0, -1], and realize that the function is outputting the absolute value of the sum over the input list. They again submit a guess: "sum of list if positive, if negative sum multiply by -1" and take the quiz.

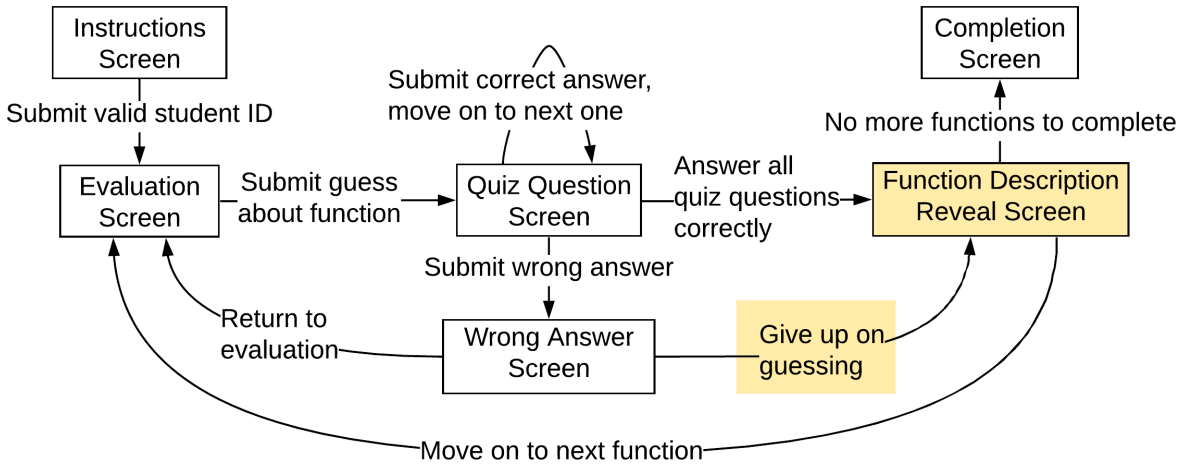
This time, Hester successfully answers all of the quiz questions, and is taken to a page revealing what the function is.



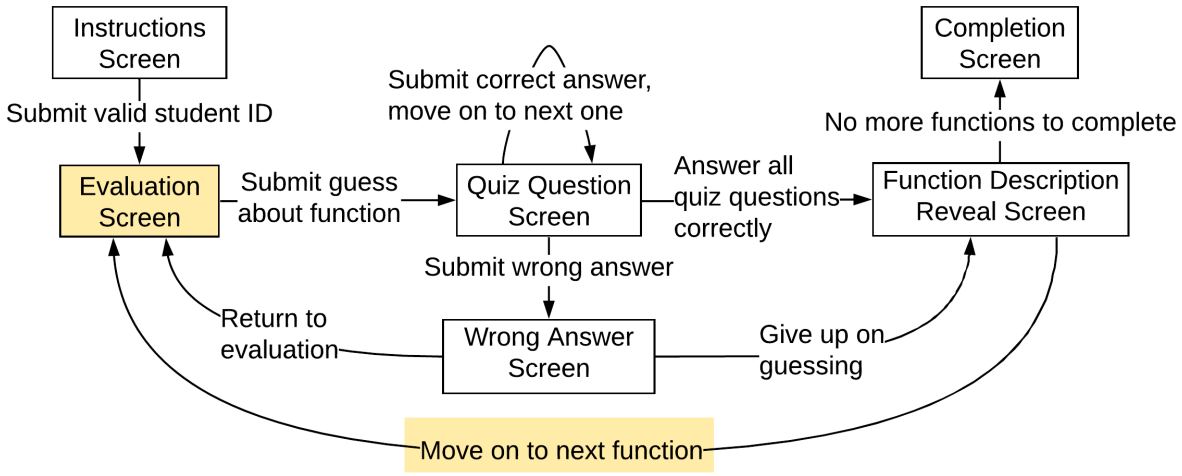
The application sends the last saved guess about the function to server as the final logged action for this mystery function.



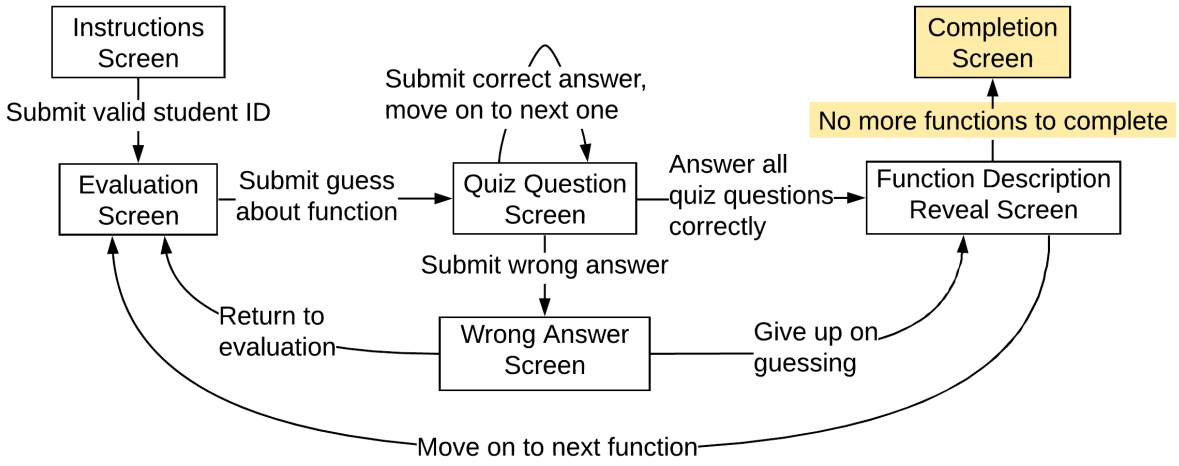
Had Hester been unable to complete the quiz and chosen to give up on this mystery function, they would have arrived at this page as well.



Hester moves on to the next page, which is the Evaluation Screen for the next mystery function.



Eventually, they complete all of the mystery functions, and finally reach a page that indicates they are done with the study.



Unlike Hester, not all respondents completed every mystery function. However, as long as the subject submitted a guess about the function, the data from that mystery function session was used for analysis.

2.1 Design Limitations

We now describe the limitations of our study design that we recognize and accept as worthwhile tradeoffs given limited time and resources.

In the oracle model of studying theory formation described in the Introduction (Section 1), the oracle can give a definite answer to the participant as to whether their guess about the phenomenon they seek to understand. Ideally, a participant in Mystery Functions is able to ascertain whether or not their guess about the function is correct without fail. We considered implementing this capability by having the students submit code that they think does the same computation as the mystery function. The web application would evaluate the submitted function on a set of inputs that we believe sufficiently cover the input domain. If their function produces the same outputs as our mystery function in an acceptable time limit, it would be deemed correct. Otherwise, it would be deemed incorrect. Alternatively, we could simply reveal what proportion of test cases were passed for any given submission.

Having participants take the quiz and submit a written answer less reliably allows us to know if the subject understands the function. If they do not answer all of the quiz questions

correctly, we can confidently say they don't understand the function, since we picked the quiz inputs to cover important areas of the input domain. However, it is not necessarily true that they do understand the function if they answer all of the quiz questions successfully. It is possible to guess what the output of the given input without actually understanding the function, especially if there are few possible values for the output type (such as booleans).

We decided to take this approach anyway, because we wanted to conduct our study with people without programming experience. As a result, our subjects are diverse in their familiarity with computing and skill at the Mystery Functions activity, which we believe is valuable for understanding theory formation at large.

2.2 Execution Limitations

We now describe improvements to our study execution that we wish we had made, as they would have strengthened its validity and potential to uncover deeper insights into theory formation.

First is the incomplete recording of guesses about functions. An improvement we could have made on this Only the last guess that a subject submits for a function before moving on to the next one is saved. Furthermore, we do not ask subjects to articulate how sure they are about their guess. Saving all submitted guesses and confidence levels would have allowed for deeper insight into why subjects chose certain inputs, and how they informed the final theories they submitted.

Another area of improvement is inter-rater reliability. In order to ensure a high level of consensus among raters of guesses, there needs to be a standardized rubric that is used to determine what labels apply to each answer. The labels we used are listed in Guess Labeling under Analysis (Section 5.1). In this project, a rubric was developed, but only one author assigned labels and did the resulting analysis. Ideally, multiple authors would assign labels and determine a high Cohen's before analysis is carried out.

3 Functions Given

The mystery functions included were chosen to represent a wide range of possible functions, and take on average 10 minutes each to complete for students to complete. They were presented in random order. Participants were presented 5 mystery functions to complete, but due to mistakes in data collection for 1 of the matched pairs, we decided not to use it for analysis in this project; we will describe and present analysis for only the other 4 functions.

A preliminary trial study run was conducted on Amazon Mechanical Turk to gauge the difficulty of two versions of a potential candidate function **IsPalindrome**. The first version of IsPalindrome takes in a list of integers, and outputs a boolean: whether the input list is palindromic or not. The second version takes in a string, and returns whether the string is palindromic. Each version of IsPalindrome was assigned to 5 Turk workers with programming experience. On both versions of the function, workers took 5 minutes or less to complete the session, but fewer than half gave correct final guesses about the function. As a result, we chose simpler functions for the study.

Below is a description of the functions used in the study.

- **Average** takes in a list of integers, and returns an integer that is the average of the input list.
- **Median** takes in a list of integers, and returns an integer that is the median of numbers in list. This is the middle number of sorted input list its length is odd; average of the two middle numbers otherwise.
- **SumParityBool** takes in a list of integers, and returns the boolean value true if the parity of list is 1; false otherwise.
- **SumParityInt** takes in a list of integers, and returns the parity of the sum of numbers in list: the remainder of sum divided by 2.
- **SumBetween** takes in two integers, and returns the sum of integers between first and second input integers, inclusive. If the inputs are equal, the result is that value; if the first is larger, the result is 0.

- **MultiplyAndHalve** takes in an integer that we will call x. It outputs $\frac{x \cdot (x-1)}{2}$.

Below is a table displaying the inputs and corresponding outputs used for the quiz questions on each function. Each input output pair is denoted by the input value followed by an arrow ($\rightarrow$) and then its output value.

	Quiz Input Output Pairs		
Function	**First Input and Output**	**Second Input and Output**	**Third Input and Output**
Median	[1, 8, 24] → 8	[8, 1, 24] → 8	[1, 2, 3, 14] → 2.5
Average	[1, 8, 24] → 11	[8, 1, 24] → 11	[1, 2, 3, 14] → 5
SumParityBool	[333] → TRUE	[-8, 3] → FALSE	[1, 0, 1] → FALSE
SumParityInt	[333] → 1	[-8, 3] → 1	[1, 0, 1] → 0
Sum Between	2, 5 → 14	5, 2 → 0	-2, 3 → 3
Induced	11 → 55	9 → 36	6 → 15

Some of the functions come in matched pairs such that every subject was only shown one of them. Upon visiting the web application URL, one of the matched pair functions is chosen uniformly at random for that subject. We wanted to see if there would be a significant difference in performance due to the pair's differences. The first matched pair is **Average** and **Median**, which perform similar but distinct calculations; the second is **SumParityBool** and **SumParityInt**, which differ only in output type.

4 Subjects

Our participants came from two distinct sources. At Indiana University, students can receive course credit by partipating in studies administered by the Department of Psychological and Brain Sciences (PBS). Students enrolled in approved PBS courses are eligible to sign up for the program, and must complete a number of 50 minute studies over the course of a semester. The functions used for the mystery functions study were chosen to take approximately 10 minutes each so that the total time taken would be approximately 50 minutes to accom-

modate this constraint. 4.3% of participants majored in an explicitly computing-related discipline in the Luddy School of Informatics, Computing, and Engineering.

At Brown University, Introduction to Software Engineering (CSCI 0320) students were given the option of completing our study for extra credit. The course involves substantial programming projects to be completed individually and also in pairs and teams of 3-4 students. During the semester that our study was administered, most students in the course took it as a requirement for the computer science major. Additionally, most of the students were 2^{nd} or 3^{rd} year undergraduates.

We administered the Indiana University portion of the study in the fall semester of the 2019-2020 school year, and the Brown portion in the spring semester of the 2019-2020 school year. From Indiana University, there were a total of 62 respondents who submitted a well-formed guess about at least one of the functions that described an expression relating inputs and outputs of the appropriate type; from Brown University there were 80.

5 Analysis

We analyzed the inputs evaluated, quiz attempts made, and written guesses submitted by subjects for 4 out of 5 functions that they were presented during the study, as outlined in Functions Given (Section 3). All percentage values in this section are rounded to the nearest ones digit.

To simplify the analysis, we treated the two versions of the study at Indiana and Brown University as the same. They are identical except for a small improvement made for the Brown version. In the fall iteration of the study at Indiana University, subjects are able to evaluate quiz inputs after they get a question wrong, thereby learning what the right answer is. This makes it easy to game the quiz, making the quiz a less effective tool for seeing if a subject understands the function. In the spring iteration of the study at Brown, we banned subjects from evaluating inputs after seeing them during a quiz attempt. We felt that this would force subjects to rely more on their ability to choose inputs and interpret feedback than on getting hints from the quiz.

5.1 Guess Labeling

We developed a rubric for classifying the final using labels that fall into the following categories: well-formedness, correctness, general, and function specific.

The well-formedness labels describe whether the guess can be interpreted as a guess about the function at all. Some guesses do not describe a function with the appropriate input and output types, or even describe a concrete expression or pattern observed among input output pairs.

Well-Formedness Labels	
NONS	Says something that can't be interpreted as a function guess, like "3" or "FALSE"
IDK	Says "I don't know" or similar and nothing else
NORM	Describes an expression relating inputs and outputs of the correct type

Out of 168 total participants from both subject pools, 100% submitted a written answer for the 1st function presented, 94% for the 2nd function, 90% for the 3rd, and 79% for the 4th. Not every answer could be interpreted as a guess about the function, describing an expression relating inputs and outputs of the appropriate types. Out of all participants, 74% submitted a well-formed guess for the 1st function presented, 73% for the 2nd function, 69% for the 3rd, and 59% for the 4th.

The correctness labels are scores between 1 and 4 inclusive—4 being the highest.

Correctness Ratings	
4	Completely or almost completely correct
3	Mostly correct
2	Somewhat correct
1	Not at all correct

The general labels describe properties of the guess that can appear across all functions, such as whether it uses code-like syntax or pseudocode to describe the function.

General Labels	
UNS	Says something comprehensible and includes a statement of uncertainty, like "I'm not sure"
NINP	Names the function parameters
NFCN	Names the function
NOUT	Names the output
IO	Lists at least one input/output pair as an example
CODE	Uses code-like syntax or pseudocode
SET	For functions that take in lists: uses the word "set" to refer to the collection of items in the input

The function specific labels below characterize traits of the expression that the guess describes that are specific to the function. Students not only described varying expressions, but also articulated the same expressions in different ways.

Average Labels	
A-WORD	Used the word "average" or "mean"
A-CALC	Describes calculation of average: sum divided by length of list
A-MED	Describes the Median function instead
Median Labels	
M-WORD	Used the word 'median'
M-CALC	Describes calculation of median: in sorted list, middle element if odd length; average of two middle ones if even length
M-WRO	Used word 'median' incorrectly
M-AVG	Describes the Average function instead
M-MID	Used the word 'middle' or 'midpoint' or similar

SumParityBool Labels	
SPB-TC	Specifies condition needed for an output of TRUE completely correctly
SPB-TP	Specifies condition needed for an output of TRUE partially correctly
SPB-TN	Specifies condition needed for an output of TRUE not at all correctly
SPB-FC	Specifies condition needed for an output of FALSE completely correctly
SPB-FP	Specifies condition needed for an output of FALSE partially correctly
SPB-FN	Specifies condition needed for an output of FALSE not at all correctly
SPB-VSUM	Mentions the parity of the sum of the input list elements
SPB-VELEM	Mentions the parity of the sum of one or only some of the list elements
SPB-VOTH	Mentions the parity of some other value
SPB-PPAR	Mentions the concept of parity using the word “parity”
SPB-PEO	Mentions the concept of parity using the words “even” and “odd”
SPB-PMOD	Mentions the concept of parity using the word “mod” or “modulo”
SPB-PREM	Mentions the concept of parity using the word “remainder” or “division”
SumParityInt Labels	
SPI-1C	Specifies condition needed for an output of 1 completely correctly
SPI-1P	Specifies condition needed for an output of 1 partially correctly
SPI-1N	Specifies condition needed for an output of 1 not at all correctly
SPI-0C	Specifies condition needed for an output of 0 completely correctly
SPI-0P	Specifies condition needed for an output of 0 partially correctly
SPI-0N	Specifies condition needed for an output of 0 not at all correctly
SPI-VSUM	Mentions the parity of the sum of the input list elements
SPI-VELEM	Mentions the parity of the sum of one or only some of the list elements
SPI-VOTH	Mentions the parity of some other value
SPI-PPAR	Mentions the concept of parity using the word “parity”
SPI-PEO	Mentions the concept of parity using the words “even” and “odd”
SPI-PMOD	Mentions the concept of parity using the word “mod” or “modulo”
SPI-PREM	Mentions the concept of parity using the word “remainder” or “division”

SumBetween Labels	
SB-PAT	Describes pattern instead of an expression that can be evaluated
SB-RBASE	Describes a recurrence relation with a base case
SB-ATH	Describes a closed form arithmetic expression. Correct is ((A+B)(B-A+1)/2) or equivalent
SB-WOR	Describes a closed form expression in words
SB-ONE	Expression describes a single sum. Correct is [A, B]
SB-TWO	Expression describes 2 distinct sums. Correct is [1, B] - [1, A-1]
SB-SASC	Describes sum; bounds are in ascending order, lower before higher
SB-SDSC	Describes sum; bounds are in descending order, higher before lower
SB-SBOUND	Describes sum; specifies whether bounds are inclusive or exclusive. Correct is both inclusive
SB-CON	Describes sum with conventional interval notation, like [A, B]
SB-WORD	Describes sum in words like “A to B” or “between A and B”
SB-SEQ	Describes sum as a sum of sequence, like A + (A+1) + ... + (B-1) + B

MultiplyAndHalve Labels	
MH-DNEG	Gives different expressions for positive and negative inputs
MH-NEG	Explicitly mentions negative inputs
MH-DIST	Describes pattern of difference between outputs of consecutive inputs
MH-MULT	Describes pattern of some value being multiplied by a factor that increases with input values
MH-WORD	Describes closed form expression in words
MH-ATH	Describes primarily with equations or symbols
MH-T	Mentions triangular numbers
MH-TN-1	Expression described as the (n-1)th triangular number
MH-TN	Expression described as the nth triangular number
MH-SASC	Describes sum; bounds are in ascending order, lower before higher
MH-SDSC	Describes sum; bounds are in descending order, higher before lower
MH-SBOUND	Describes sum; specifies whether bounds are inclusive or exclusive. Correct is both inclusive
MH-SWORD	Describes sum in words like "A to B" or "between A and B"
MH-SDSEQ	Describes sum as a sum of a sequence like: A + (A+1) + ... + (B-1) + B
MH-SCON	Describes sum with conventional interval notation, like [A, B]
MH-R1ST	Describes recurrence relation of f(x) = f(x-1) + (x-1) or equivalent
MH-FIB	Mentions the fibonacci sequence
MH-R2ND	Describes recurrence relation of f(x) = f(x-1) + f(x-1) or equivalent
MH-RBASE	Describes a recurrence relation with a base case

5.2 Correctness Ratings

Each function guess was given a correctness score between 1 to 4 inclusive, 4 being the highest score. We acknowledge that because the rater of the guesses is also a Brown student, there is potential for bias in the ratings given despite the standardized system for assigning labels and scores.

Below are the proportions of students who received each correctness rating, by function and subject pool. Each percentage value for a correctness rating should be interpreted as the proportion of well-formed guesses per function and subject pool that received that particular correctness rating. Each "Well-Formed Guesses" percentage value is the proportion of all submitted guesses for that function and subject pool that were well-formed.

Average Function					
Subject Pool	Not at all correct	Somewhat correct	Mostly correct	Completely correct	Well-Formed Guesses
Indiana	12%	0%	15%	73%	63%
Brown	0%	7%	7%	87%	100%
Median Function					
Subject Pool	Not at all correct	Somewhat correct	Mostly correct	Completely correct	Well-Formed Guesses
Indiana	27%	15%	42%	15%	67%
Brown	2%	20%	7%	70%	98%

SumParityBool Function					
Subject Pool	Not at all correct	Somewhat correct	Mostly correct	Completely correct	Well-Formed Guesses
Indiana	32%	16%	37%	16%	58%
Brown	5%	5%	8%	82%	100%
SumParityInt Function					
Subject Pool	Not at all correct	Somewhat correct	Mostly correct	Completely correct	Well-Formed Guesses
Indiana	8%	25%	54%	13%	53%
Brown	0%	14%	3%	83%	97%

SumBetween Function					
Subject Pool	Not at all correct	Somewhat correct	Mostly correct	Completely correct	Well-Formed Guesses
Indiana	22%	58%	19%	0%	47%
Brown	3%	15%	25%	57%	91%

MultiplyAndHalve Function					
Subject Pool	Not at all correct	Somewhat correct	Mostly correct	Completely correct	Well-Formed Guesses
Indiana	49%	13%	21%	17%	57%
Brown	0%	8%	46%	45%	99%

5.3 Differences in Matched Pair Function Difficulty

We now discuss the difference in correctness ratings between the Average and Median functions by subject pool. 87% of Brown students guessed Average completely correctly, while only 70% guessed Median completely correctly. 73% of Indiana students guessed Average completely correctly, while only 15% guessed Median completely correctly.

Furthermore, students more often mistook the Median function for the Average function (as denoted by label M-AVG) than vice versa (as denoted by label A-MED). 7% of Brown students mistook the Average function for the Median function, while 16% of them mistook Median for Average. 4% of Indiana students mistook the Average function for the Median function, while 23% of them mistook Median for Average.

We conclude that Median is harder to guess than Average.

We now discuss the difference in correctness ratings between the SumParityBool and SumParityInt functions by subject pool. The proportions of students who guessed each function completely correctly are similar: 16% of Indiana students for SumParityBool and 13% for SumParityInt; 82% of Brown students for SumParityBool and 83% for SumParityInt. However, more students in both populations gave not at all or somewhat correct guesses for SumParityBool than SumParityInt. For example, only 8% of Indiana students

guessed SumParityInt completely incorrectly, while 32% guessed SumParityBool completely incorrectly.

We conclude that SumParityBool is harder to guess than SumParityInt.

5.4 Improvement Over Time

Across all subjects, the average correctness score that a subject gets on a function doesn't appear to correlate with whether the subject saw the function earlier or later. The average scores of those who completed two functions are 2.48 and 2.29; those who completed three functions had average scores of 3.15, 2.90, and 3.08; those who completed all four had average scores of 3.39, 3.25, 3.25, and 3.20. We do not see evidence of better performance on later functions seen.

5.5 Inputs Evaluated Before and After Quiz Attempts

In both groups, subjects who scored the lowest and the highest—receiving scores 1 or 4—made the fewest quiz attempts on average across all functions. 67% of Indiana students and 86% of Brown students only attempted the quiz once.

Indiana students who made multiple attempts evaluated 62% of all their inputs before the first ever quiz attempt; for Brown students, 61%.

We conclude that subjects who submitted better quality guesses about functions were not able to do so primarily because they made multiple quiz attempts and received more feedback on the correctness of their guesses as well as ideas for what inputs to evaluate.

5.6 Choosing Inputs

To investigate how systematically subjects chose inputs to evaluate, we examined how similar consecutive inputs evaluated were.

We chose to characterize the change between inputs evaluated in terms of the minimum number and kind of edit operations needed to convert from one to the other. For lists of integers, we define an edit operation as an insertion, deletion, or replacement of one of the integers at a valid index of the 0-indexed list. For example, an insertion of 1 at index 0

of the empty list [] would result in [1]; a deletion at index 0 of [4, 3] results in [3], and a replacement at index 1 of [2, 4] to 5 results in [2, 5].

For our analysis, we did not consider the list indices which edit operations were applied. Instead, we considered any sequences of operations that contained the same numbers of each kind of edit operation the same. This way, we can compare the similarity of minimum operations groups between two pairs of consecutive inputs, even if the source inputs are differing lengths. For example, changing [1, 2] to [1, 7, 3, 4] at minimum requires 2 insertions and 1 replacement. Changing [2] to [8, 5, 6] requires at minimum 2 insertions 1 replacement as well.

An important consequence of ignoring the indices of operations is that any two operations groups with 2 insertions and 1 replacement are considered equivalent, even if they would result in different lists when applied to the same base list. For example, we consider operations groups with the same number of each kind of operation but different orders of application equivalent.

Also note that the minimum number and kind of edit operations is distinct from the actual character by character changes that a subject makes to the text for a past input evaluated to generate the next input to evaluate. We do not know what the actual changes and order of changes to the input text box the subject actually makes—only the resulting next input.

Under this definition of change between inputs, we define operation chains as sequences of inputs for which any two consecutive inputs is connected by the same group of operations. In other words, to generate that sequence of inputs, we would start with the first input evaluated, and apply operations groups with the same number of each kind of edit operation to each successive input until the last input has been generated. An operation chain of length 1 consists of 1 input generated with an operations group that is not repeated again; an operation chain of length 2 consists of 2 consecutive inputs, each generated by applying the same operation group to its predecessor. We can think of the sequence of all inputs evaluated by a subject for some function as a sequence of operation chains of varying length.

Below are the operation groups that make up 80% of operation chains across functions with list of integer inputs.

Operation groups by edit operation frequencies				
Replace	Insert	Delete	Proportion of all chains formed with this group	Average number of inputs in operation chain
1	0	0	31%	1.43
0	1	0	18%	1.74
2	0	0	10%	2.35
1	1	0	5%	3.65
0	0	1	4%	1.94
1	0	1	3%	2.14
0	2	0	3%	1.55
3	0	0	3%	2.03
1	0	2	2%	1.60

The operations group for most chains usually does not consist of many operations total, with 77% of of chains having groups with only 2 operations between consecutive inputs. The overall average chain length is 1.98, meaning that on average there are about 2 repeated operations groups in a row before the next input evaluated is created using a different operation group.

Though the average chain length is low, the variation in chain lengths overall and between consecutive chains is high. The average difference of the length of any chain from the average chain length is 1.8, and the average difference between the lengths of consecutive input chains is 2.4.

Furthermore, long and short operation chains are often interleaved. For every chain that is more than 1 input long, there is a 78% that all adjacent chains are of length 1. Such multiple length chains adjacent to only chains of length 1 are on average of length 4.5.

The conclusion we draw is that people tend to generate multiple inputs in a row using a consistent methodology, as evidenced by the operation chains. Even though subjects may not think about their methods of choosing inputs in terms of edit operations, we infer that they employ systematic methods to choose inputs, since they tend to evaluate long sequences

of similar inputs.

6 Future Work: Mystery Predicates

In addition to the Brown and Indiana University studies, we also designed and administered a study specifically for Brown students who took Logic for Systems (CSCI 1950Y), an upper level computer science class that serves as an introduction to formal methods. Most students are 2^{nd} or 3^{rd} year undergraduates.

Over the course of CSCI 1950Y, students learn to use model finders, which are tools to find concrete structures that satisfy sets of constraints expressed as logical formulas. They are used extensively in software and hardware verification, as well as program synthesis. Students practice writing not only constraint predicates, but also concrete instances of structures.

As a result, it is feasible to have them guess Mystery *Predicates* that take in a concrete structure, and output a single boolean value. The input concrete structure represents a graph, consisting of a set named **Node** that contains uniquely named nodes, and a binary relation named **Edges** of ordered pairs of nodes that represent edges. To construct an input to evaluate one of our mystery predicates on, students specify what nodes exist in the Node set, and what edges exist in the Edges binary relation.

Besides the change of input type, Mystery Predicates is identical to Mystery Functions: the interfaces for input evaluation, guessing the predicate, and quiz attempts are the same. We administered the mystery predicates study during the spring semester of the 2019-2020 school year, and have the resulting dataset; we did not analyze it over the course of this project.

7 Acknowledgements

A huge thank you to my advisor Tim Nelson, reader Shriram Krishnamurthi, and collaborator Rob Goldstone. It's been such a privilege working with you all. I've learned a great deal during this project, not only about conducting studies in this area of research, but also about committing to pushing an idea out into the real world and staying excited about it day in

and day out.

References

[1] DeLosh, E. L., Busemeyer, J. R., and McDaniel, M. A. Extrapolation: the sine qua non for abstraction in function learning. *Journal of Experimental Psychology. Learning, memory, and cognition 23*, 4 (Jul 1997), 968–86.

[2] Juslin, P., Olsson, H., and Olsson, A.-C. Exemplar effects in categorization and multiple-cue judgment. *Journal of Experimental Psychology. General 132*, 1 (Mar 2003), 133–56.

[3] Kalish, M. L., Lewandowsky, S., and Kruschke, J. K. Population of linear experts: knowledge partitioning and function learning. *Psychological Review 111*, 4 (Oct 2004), 1072–1099.

[4] Markant, D. B., and Gureckis, T. M. Is it better to select or to receive? learning via active and passive hypothesis testing. *Journal of Experimental Psychology. General 143*, 1 (Feb 2014), 94–122.

[5] McDaniel, M. A., and Busemeyer, J. R. The conceptual basis of function learning and extrapolation: comparison of rule-based and associative-based models. *Psychonomic Bulletin and Review 12*, 1 (Feb 2005), 24–42.

[6] Nelson, J. D., and Movellan, J. R. Active inference in concept learning. In *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 45–51.

[7] Oaksford, M., and Chater, N. A rational analysis of the selection task as optimal data selection. *Psychological Review 101* (10 1994), 608–631.