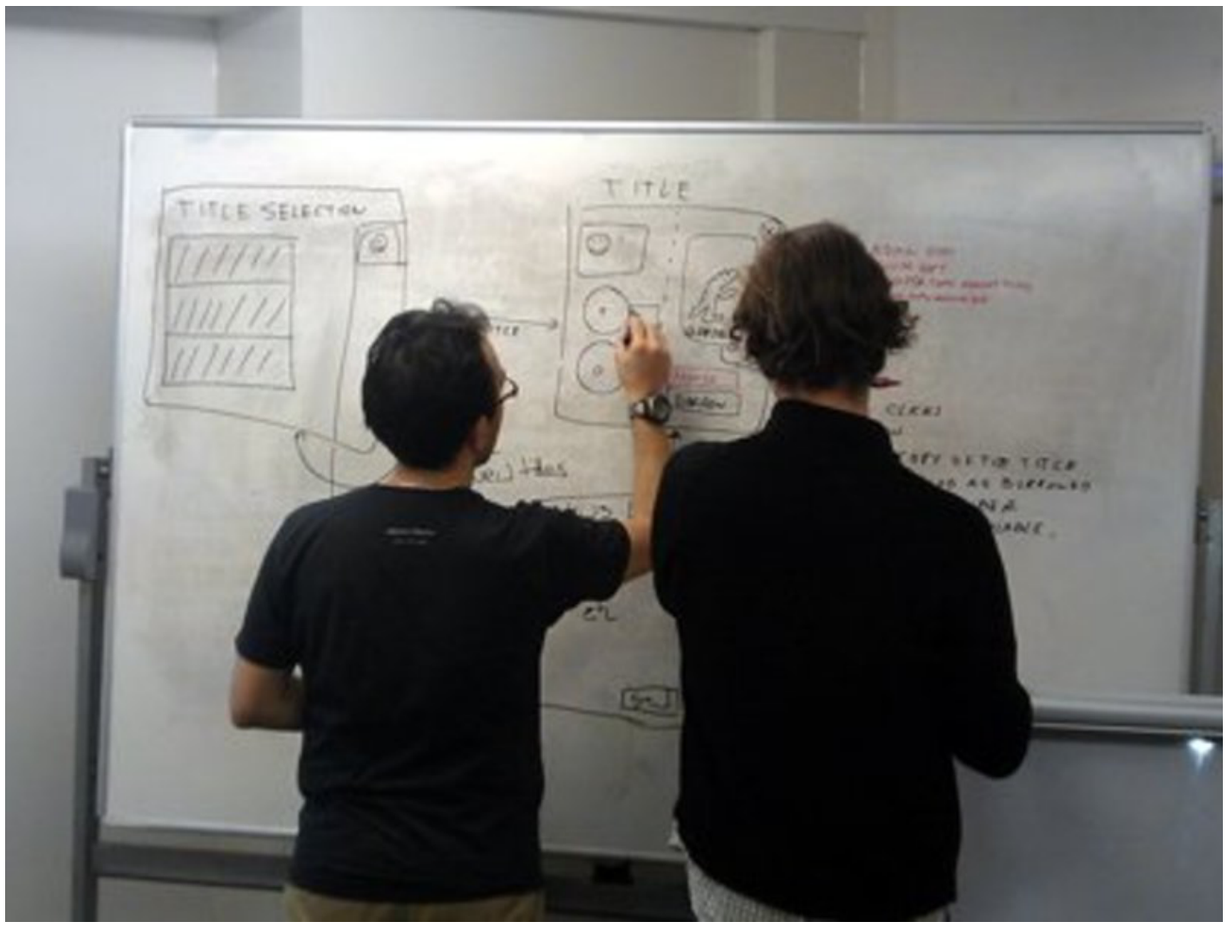
HANDWRITTEN SOURCE CODE RECOGNITION FOR TECHNICAL INTERVIEW PREPARATION

Solan Manivannan
MEng Mathematics and Computer Science

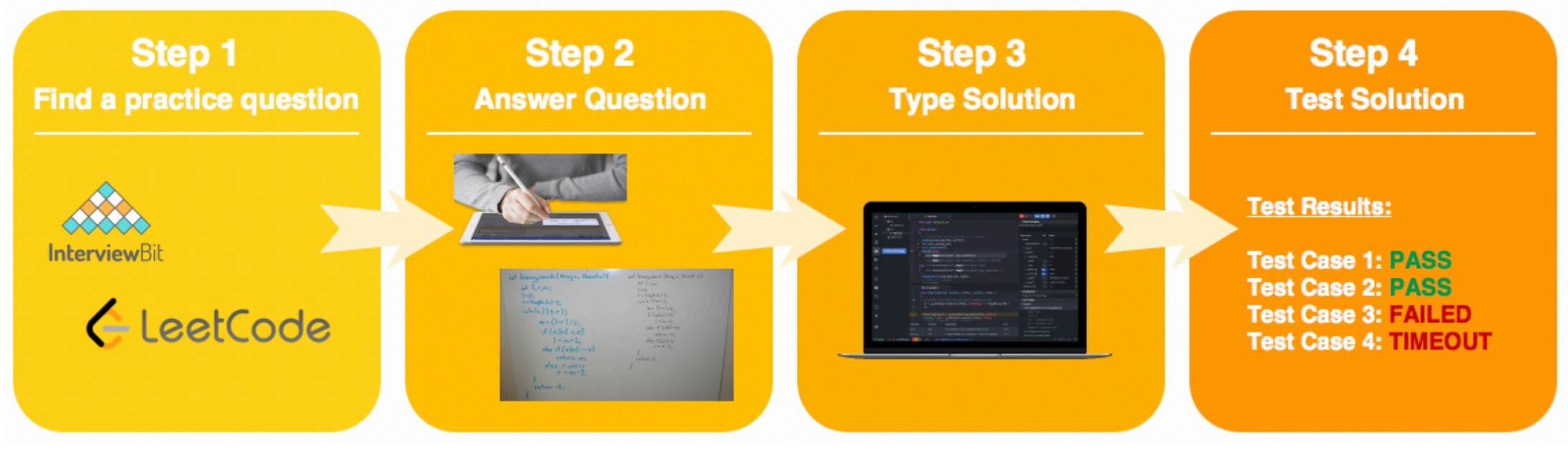
INTRODUCTION – SITUATION

facebook LEAP MOTION UBER Google jet intel amazon

Bank of America Pinterest CISCO stripe



INTRODUCTION – CORE PROBLEM



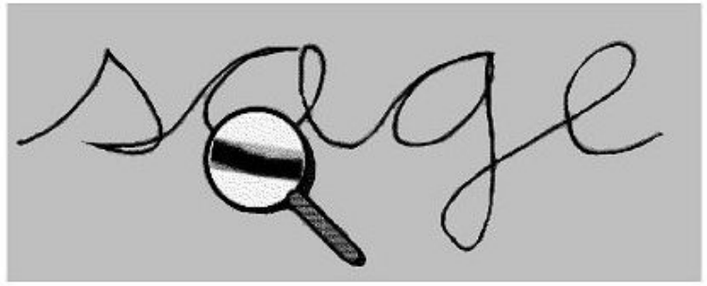
➤ Example Question:

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

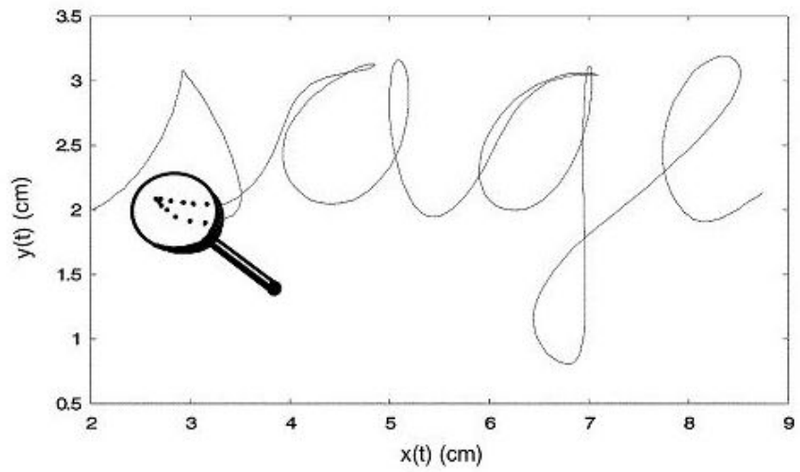
WHY CAN'T HANDWRITING RECOGNITION SYSTEMS WORK ON SOURCE CODE?

- Systems use machine learning techniques.
- Trained on large English language datasets and mathematical expressions datasets.
- Source code has
  - a mix of letters, numbers, symbols and operators.
  - structure (different between languages)
  - style (different between people)

BACKGROUND



Offline: input is treated as an image.

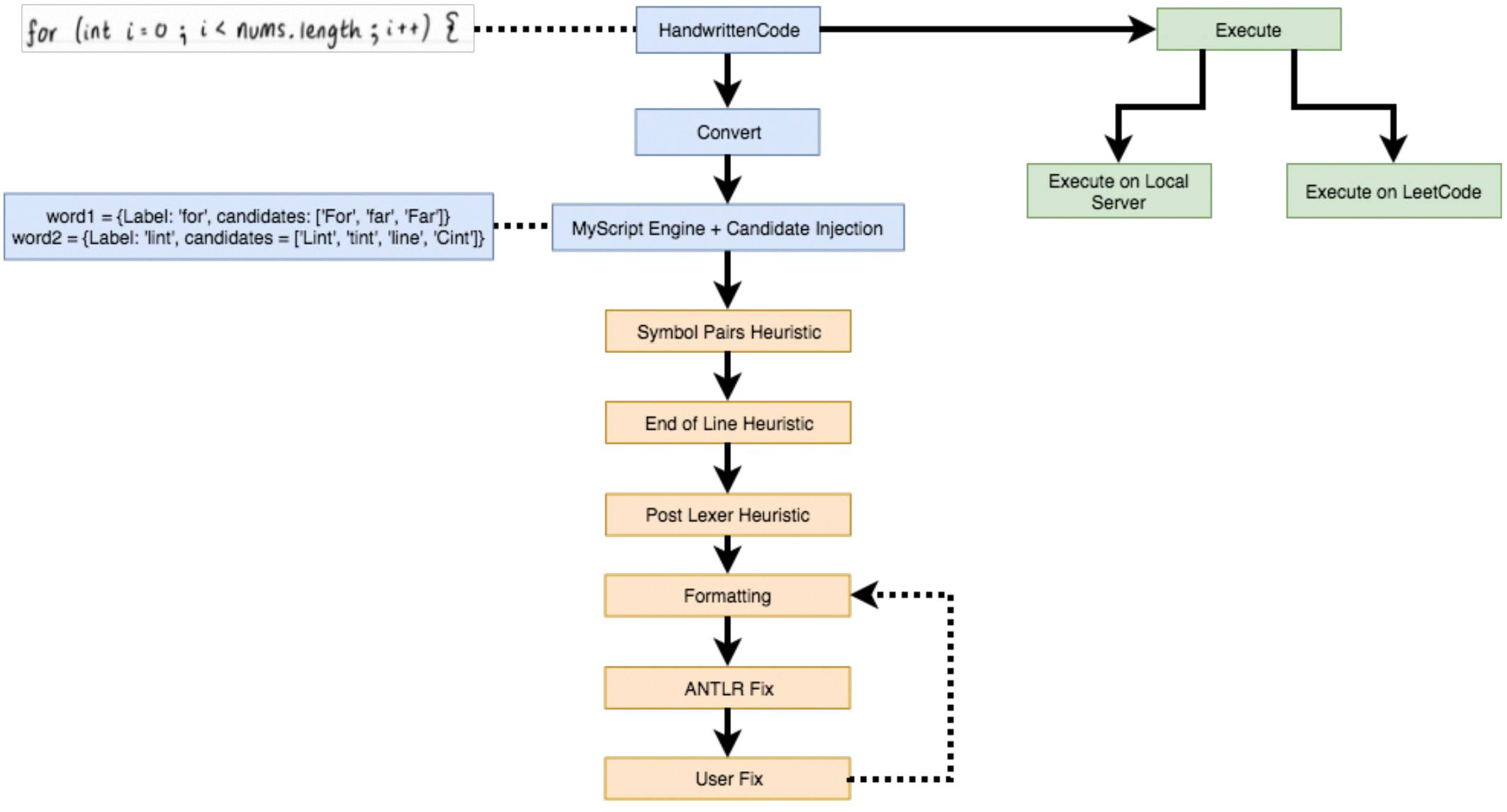


Online: the x, y coordinate is recorded as a function of time t.

- Offline vs Online Recognition
- Printed vs Cursive vs Mixed
- Variation in stroke numbers, order, shape, size and tilting angle.
- Preprocessing: thresholding, noise removal and segmentation.

MY PROPOSED SOLUTION

- Unmodified MyScript Engine + Post Processing Pipeline.
- Focus on Java programming language



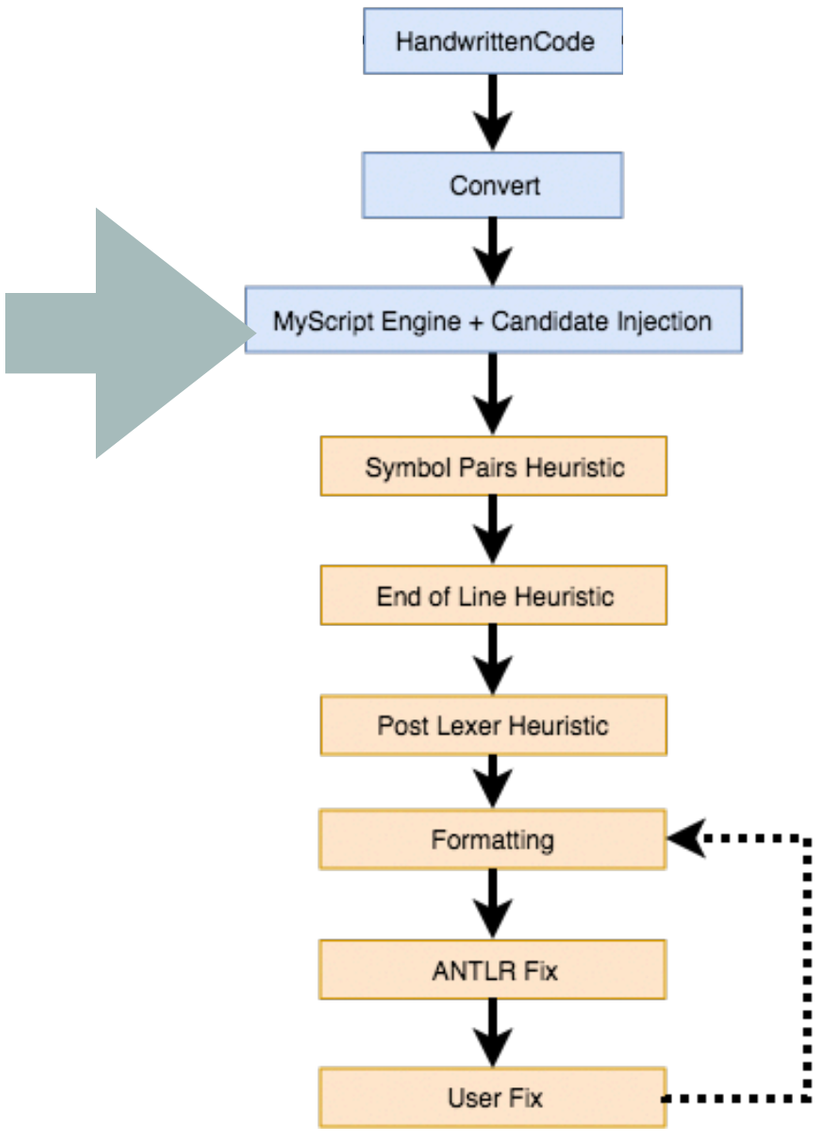
MY CONTRIBUTIONS – IOS APPLICATION

- Post Processing Pipeline
- Touch friendly UI for manual user fixes
- Compilation and Execution on local server
- Third party integration (LeetCode)
- Loading and Saving digital ink files with recognition fixes
- Generation of handwritten code samples using my dataset.

STRATEGIES FOR FIXES

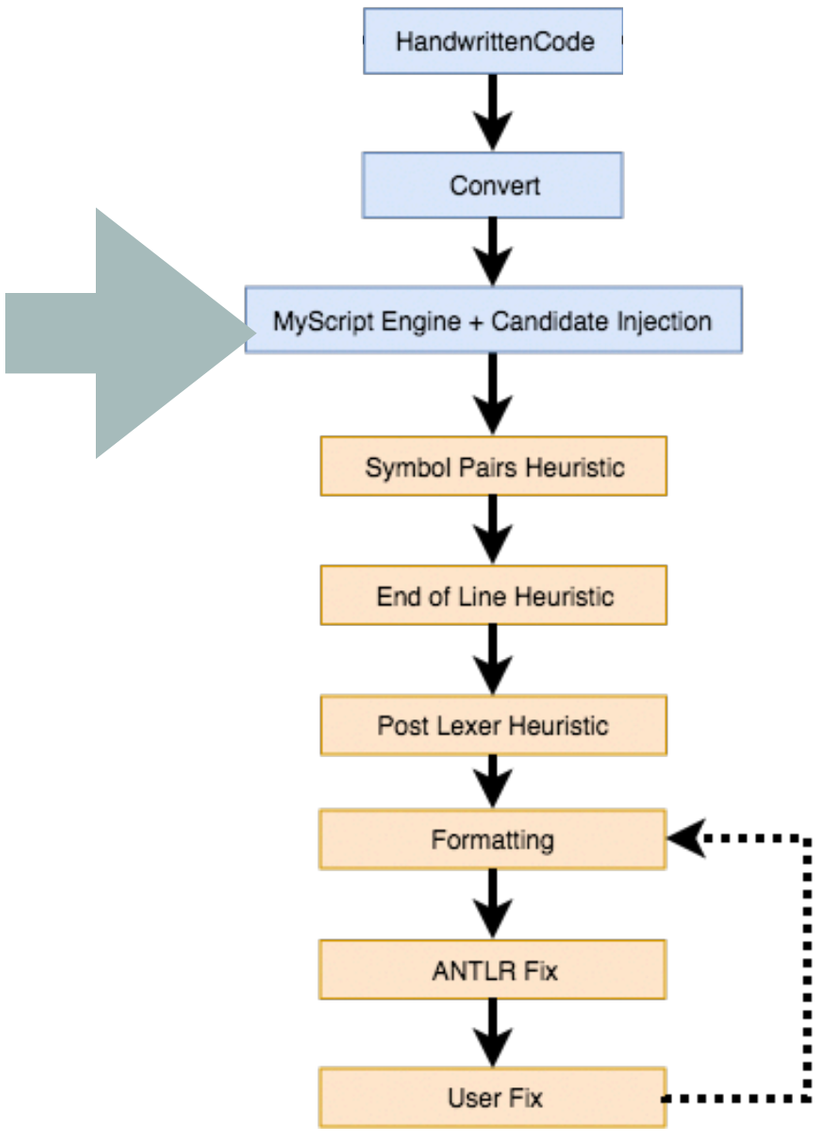
- Indicate to the user:
  - Which stage made the change
  - Old Label / New Label
- Navigate using:
  - Bounding Boxes
  - String
  - Tokens
  - Parse Tree

PIPELINE 1: CANDIDATE INJECTION



- MyScript engine provides attribute information for the contents of each bounding box.
  - Label, Alternative Candidates, Bounding Box Info (coordinates, width, height).
    - Exposed by exporting to JIIX (JSON Interactive Ink Exchange) string.
- Inject more candidates based on label matching patterns.
- Navigation: Word Object

PIPELINE 1: CANDIDATE INJECTION

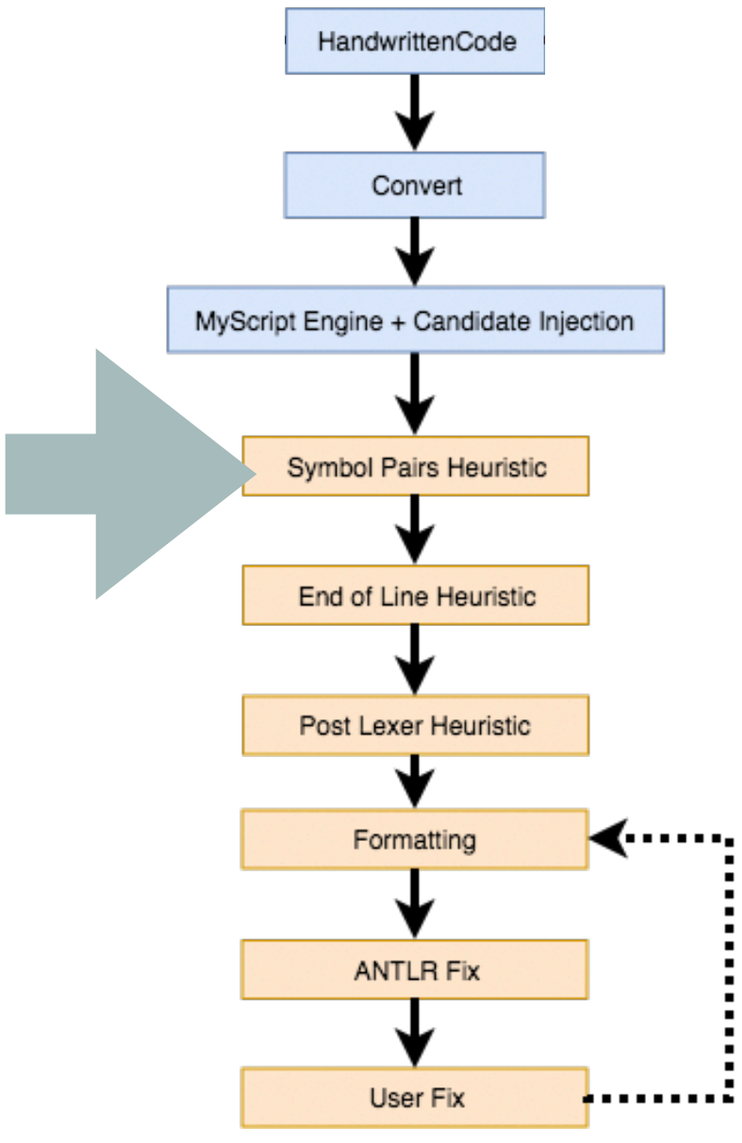


- Example:

Candidate Injection List	
Pattern	Replacement
‘C’	[“(“]
‘D’	[“1)”, “))”]

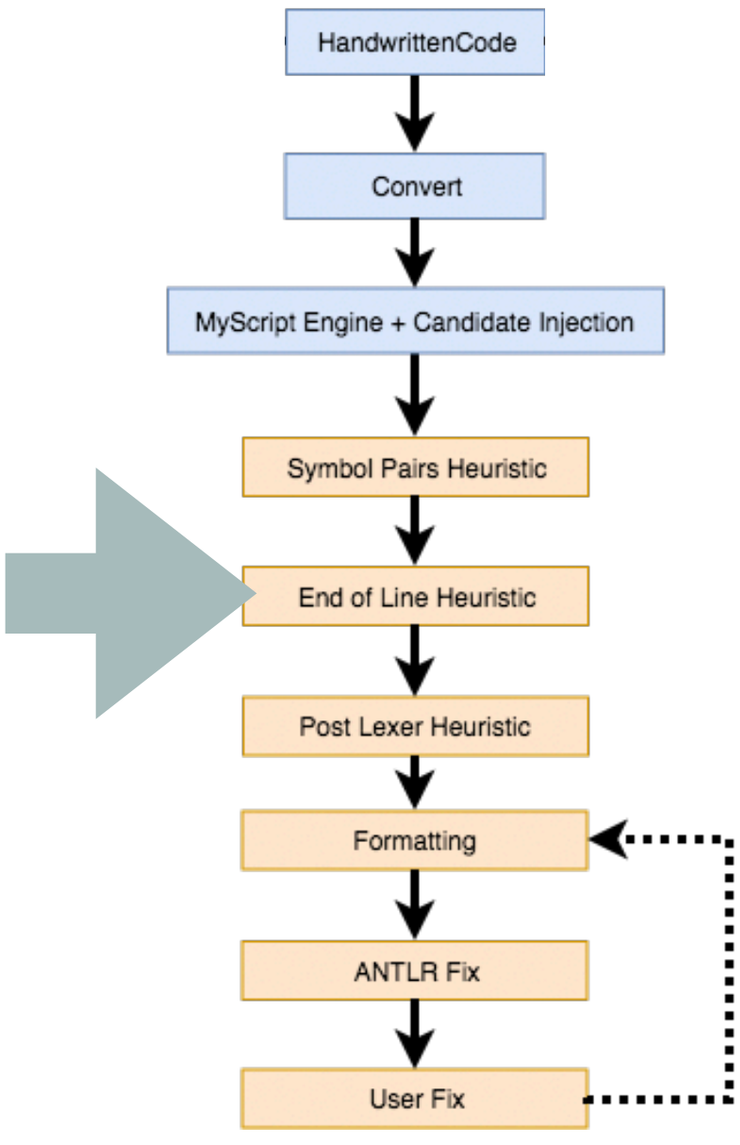
- for Cint i = 0; i < 5; i++) {

PIPELINE 2: MISMATCHING SYMBOL PAIRS



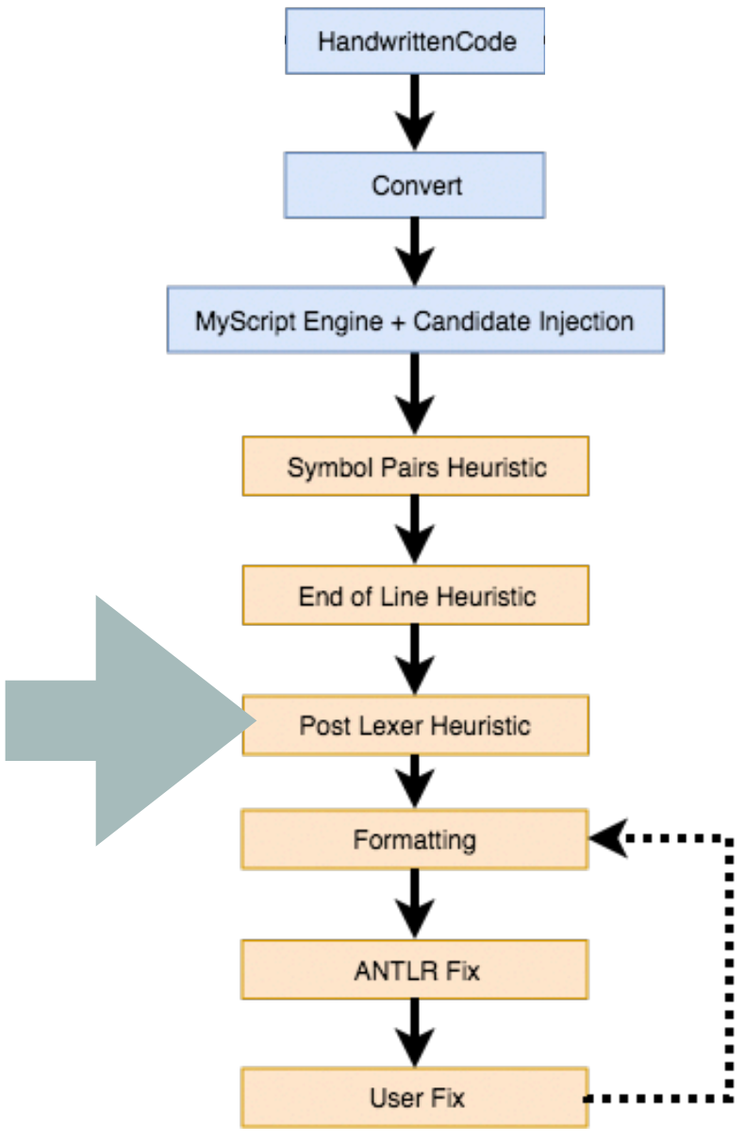
- () and [] come in complete pairs on a single line. (not <>)
- Breaks down when:
  - Overflow statement onto multiple lines.
  - Anonymous inner class
- Count opening/closing, if mismatch count, then look for missing symbol in alternative candidates, if one potential fix, then use that.
- Navigation: Word Object

PIPELINE 3: END OF LINE HEURISTIC

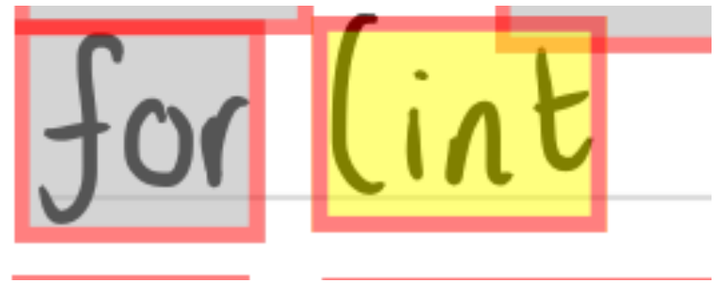


- Last character on a line should be one of { } ;
- If line has only one character, then it has to be }
- Breaks down on:
  - User adopting different code style
  - Overflowing statement
  - One-statement “if” blocks
- Navigation: Word Object

PIPELINE 4: POST LEXER HEURISTIC

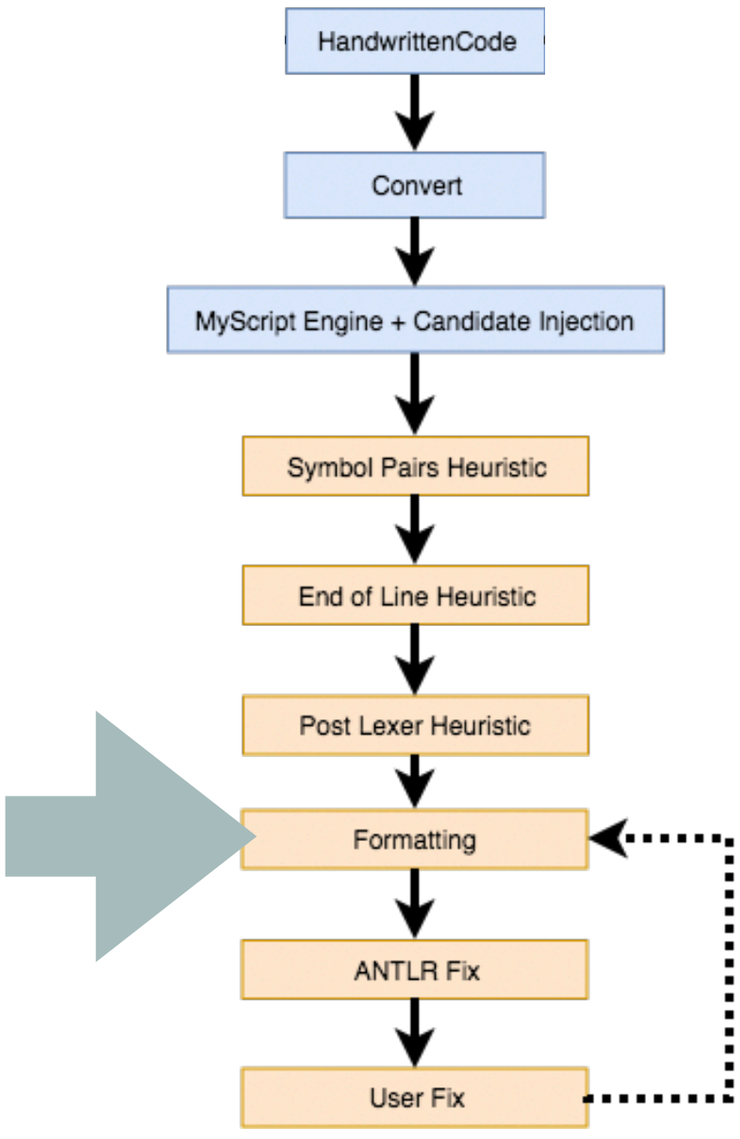


- Some tokens have a compulsory preceding/succeeding token.
- Example:
  - if (
  - } else



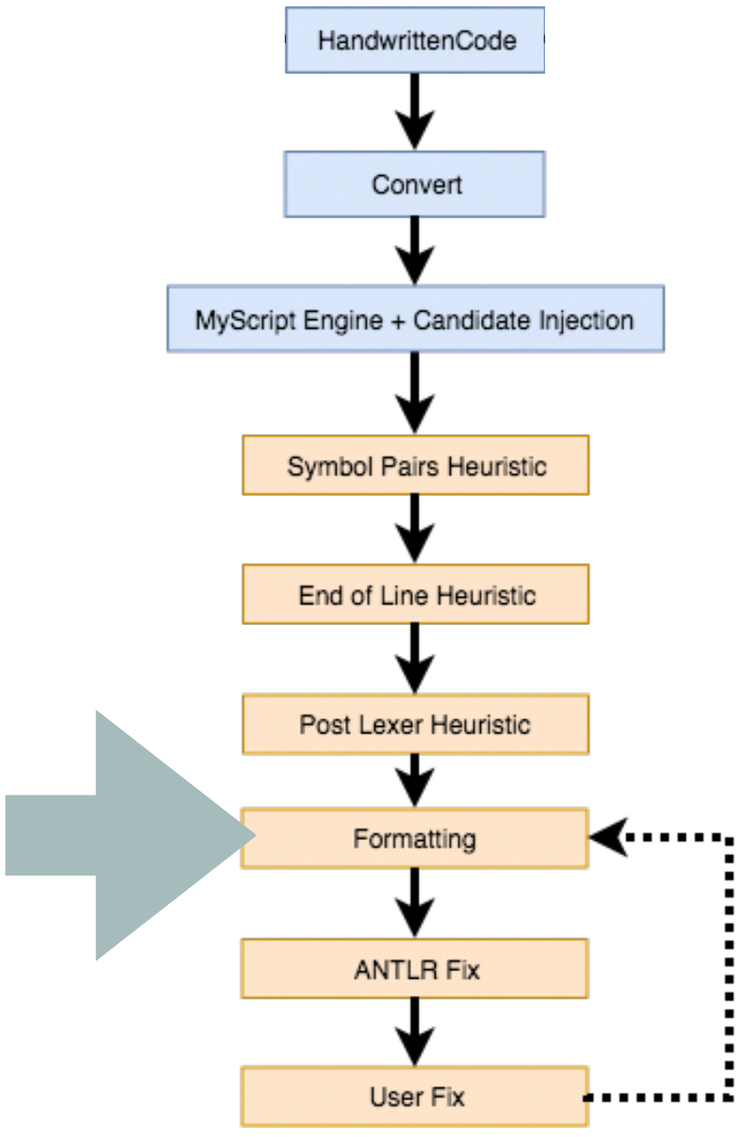
- Does not break down
- Navigation: Tokens
  - Alternatively handle in ANTLR parser/visitor to replace token.
  - Parser (time consuming)
  - Visitor (error nodes)

PIPELINE 5: FORMATTING



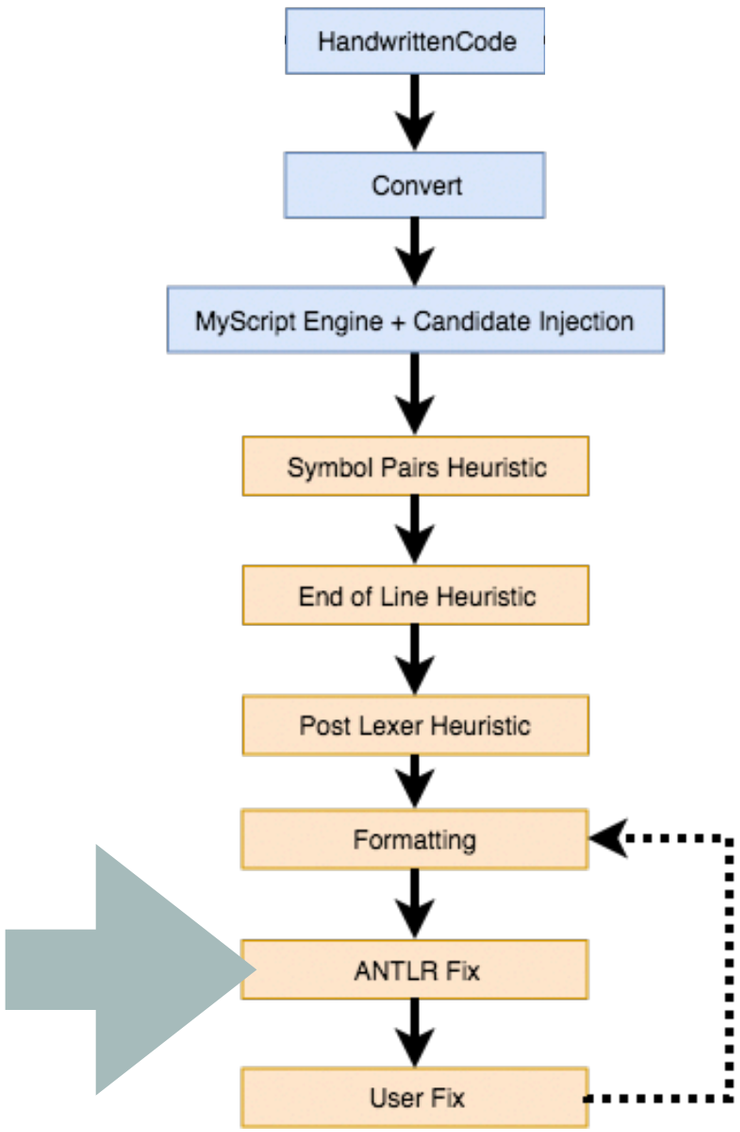
- Two common recognition errors in MyScript output:
  - Spacing (added/removed)
  - Camel casing (space added)
- Instead of identifying issues and fixing them, I 'redo' the spacing.
- Join identifier tokens that do not start with String.
- Remove spacing before '[' and between '+' '=' and all other combinations.
- Breaks down on new objects.
- Navigation: Tokens (but no UI indication)

PIPELINE 5: FORMATTING



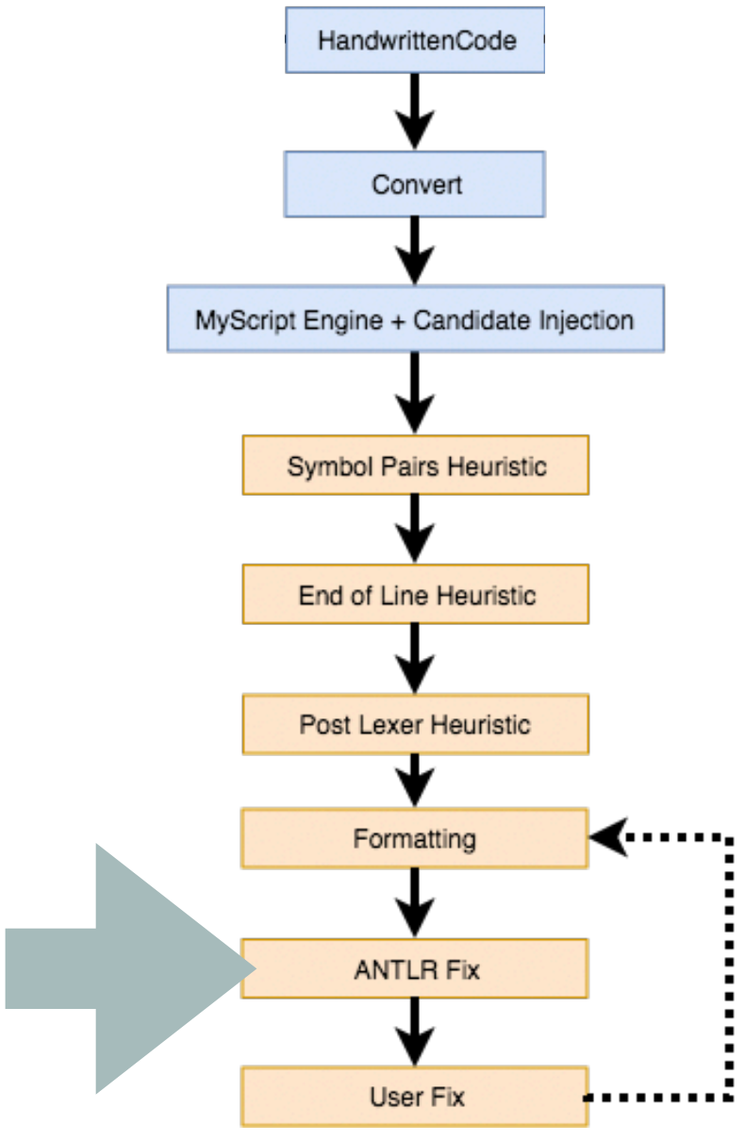
- Examples:
- “Hash Map” -> “HashMap”
  - <Identifier>, <Identifier>
- “arr [2]” -> “arr[2]”
  - <Identifier>, <LBRACK>
- “p! = 1” -> “p != 1”
  - <Identifier>, <BANG>, <EQUALS>,
- “Node List a” -> “NodeLista”
  - <Identifier>, <Identifier>, <Identifier>

PIPELINE 6: ANTLR

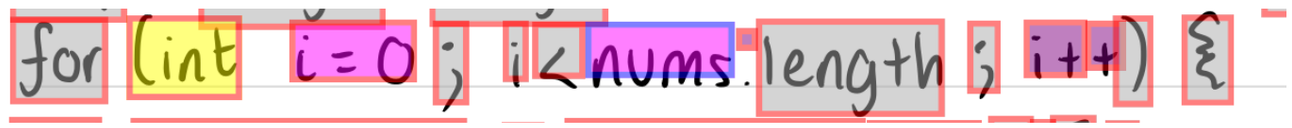


- ANTLR4 runtime (Swift) added to project.
- Track identifiers in symbol table data structure
- Fix repeat identifiers using Levenshtein distance.
- Breaks down on errors in source code that create “Error Nodes” in parse tree.
- Navigation: Parse Tree

PIPELINE 6: ANTLR

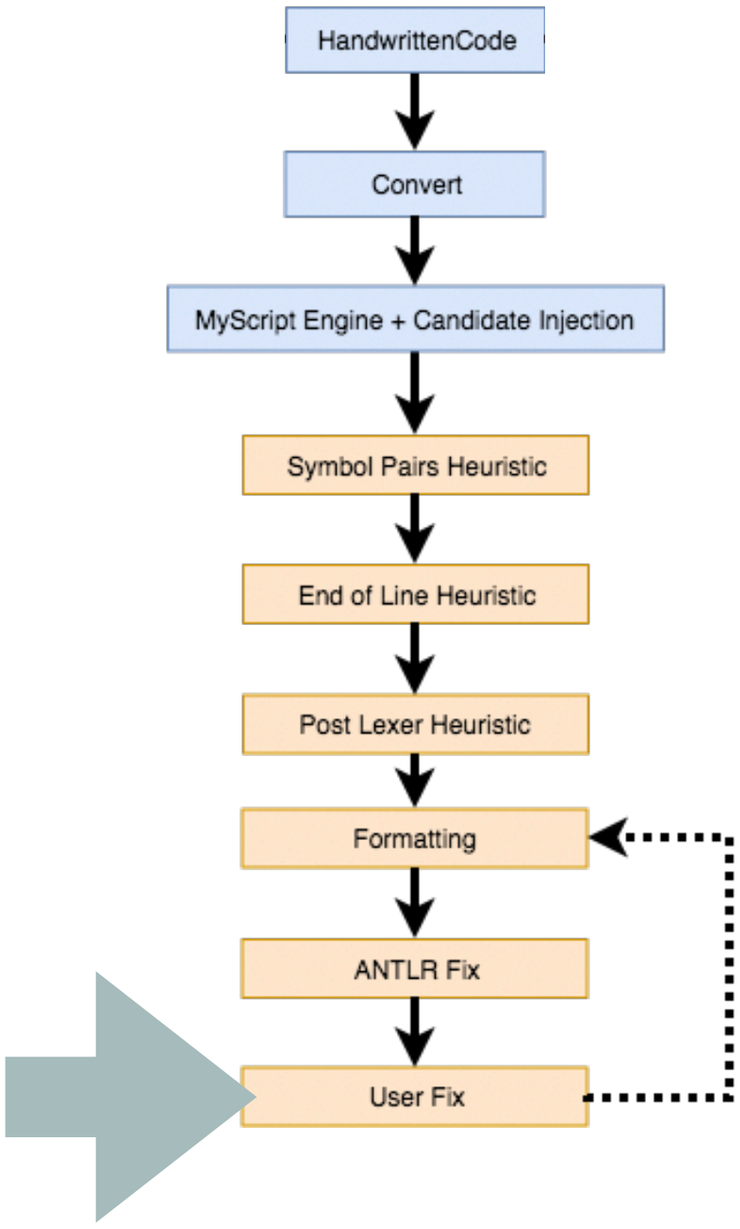


- Examples:



- Key Decisions on Levenshtein Distance:
  - Strings with same length
  - Case Insensitive
  - Threshold 65% for three letter identifiers.

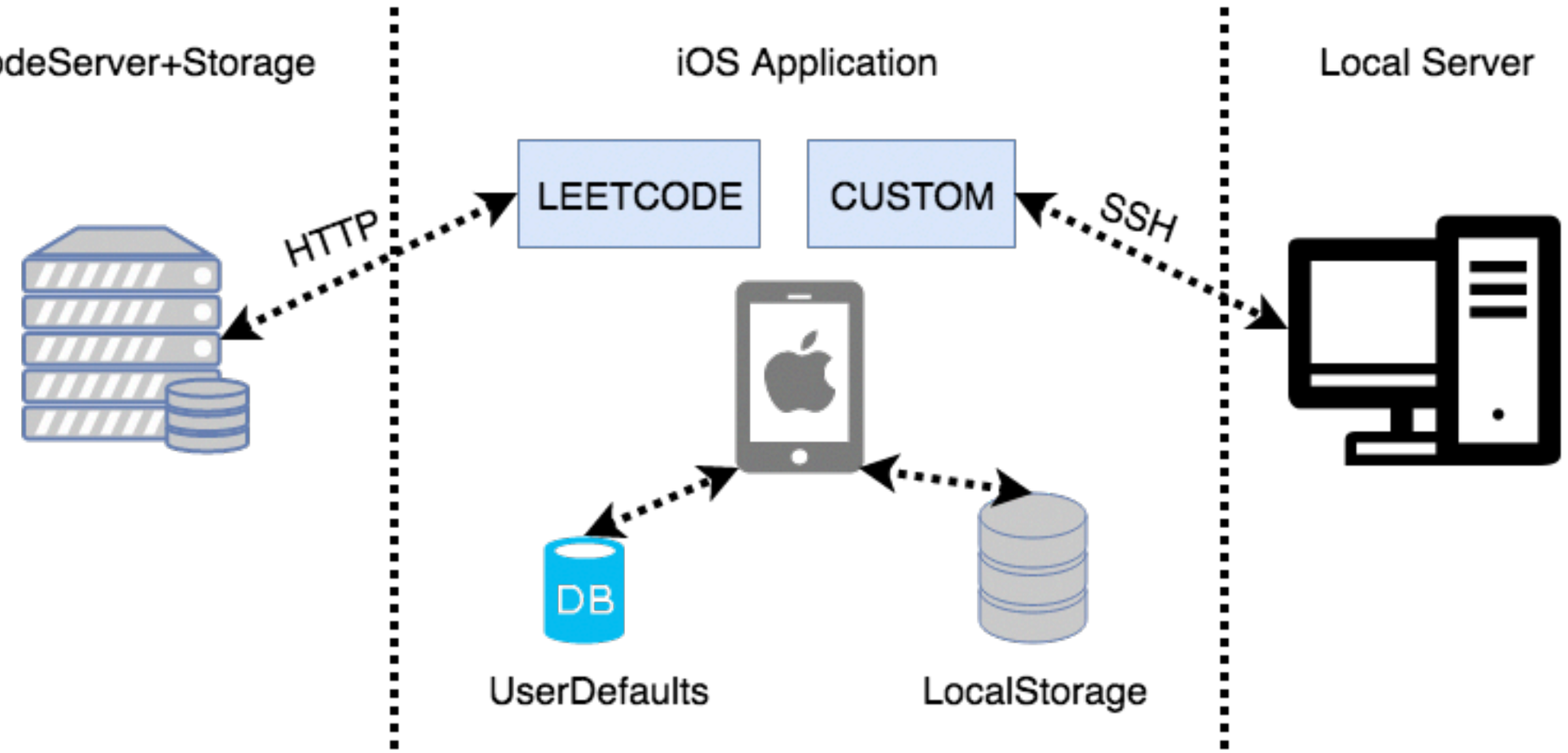
PIPELINE 7: USER FIXES



- Tapping on a bounding box allows the user to make manual fixes
- Select from alternative candidates list, add/delete space, or enter in textfield.
- Triggers formatting since syntax highlighted code needs updating, and triggers ANTLR fix again since less error nodes will make visitor more effective.
- Navigation: Word Object

DEMO

- isPrime (Local Example)
- TwoSum (LeetCode Example)



EVALUATION

- 44 handwritten source code samples of 4 LeetCode solutions.
- Assessment:
  - reduction in number of user fixes
  - reduction in time taken to get final source code.
- Change something correct to incorrect.
  - Mismatching Symbols e.g. replace < with (
  - ANTLR Levenshtein distance: MAX_VALUE / MIN_VALUE
- Use parse tree to classify each line of code, then make fixes outside of visitor class (to avoid Error nodes)

EVALUATION – REDUCTION IN USER FIXES

LeetCode Q1: Two Sum			
	Number of Fixes Required	Plain MyScript Fixes	Percentage Reduction
Solan	6	19	68.42
Generated Handwriting	12	19	36.84
10 User Average	9	21	57.14

LeetCode Q3: Longest Substring Without Repeating Characters			
	Number of Fixes Required	Plain MyScript Fixes	Percentage Reduction
Solan	12	22	45.45
Generated Handwriting	14	26	46.15
10 User Average	12	29	58.62

LeetCode Q9: Palindrome Number			
	Number of Fixes Required	Plain MyScript Fixes	Percentage Reduction
Solan	4	21	80.95
Generated Handwriting	11	19	42.11
10 User Average	10	25	60

LeetCode Q11: Container With Most Water			
	Number of Fixes Required	Plain MyScript Fixes	Percentage Reduction
Solan	11	26	57.69
Generated Handwriting	13	23	43.48
10 User Average	10	33	69.69

EVALUATION – REDUCTION IN TIME

	Average Typing Time (seconds)	Average Manual Fixing Time on App (seconds)
LeetCode Q1: Two Sum	132	66
LeetCode Q3: Longest Substring Without Repeating Characters	105	48
LeetCode Q9: Palindrome Number	186	55
LeetCode Q11: Container With Most Water	72	35

CONCLUSION

- Post processing pipeline gave good results with room for improvement.
- Hardest problem is fixing recognition of closing brackets),]
- New Problem: Gibberish code that compiles and executes, i.e. fixing errors caused by user
- Reinforcement Learning
  - Textfield user fixes create possible injection candidates
  - Frequent identifiers kept in constant symbol table.
- Machine Learning
  - Access to stroke data via export into JIIX string
  - Process each character to identify symbols without context.

EXTRA SLIDES

- Extra slides to follow

IMPLEMENTATION

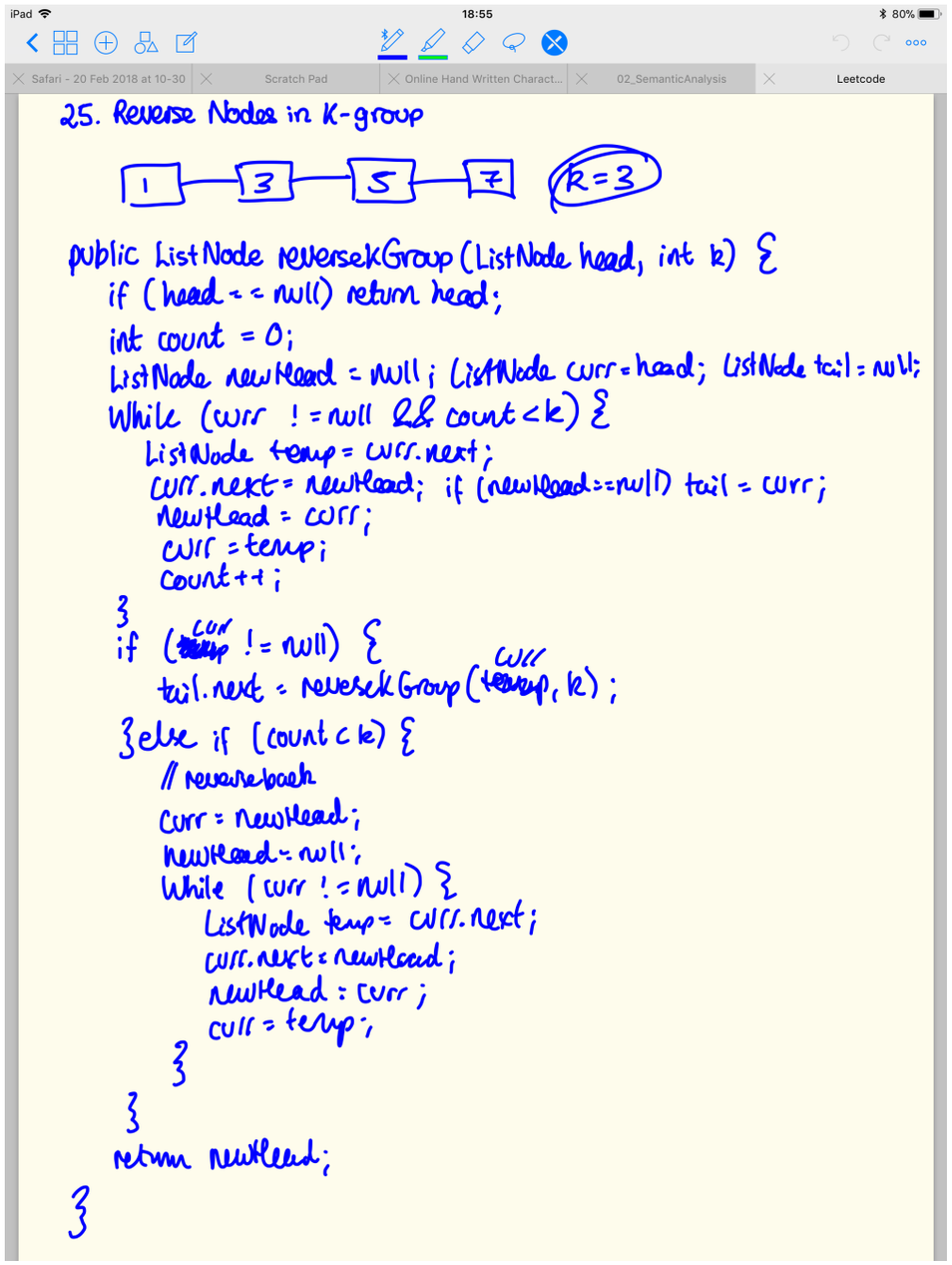
- Pipeline: demo changes
- Execution:
- DEMO custom code: isPrime with test cases
- DEMO leetcode submission

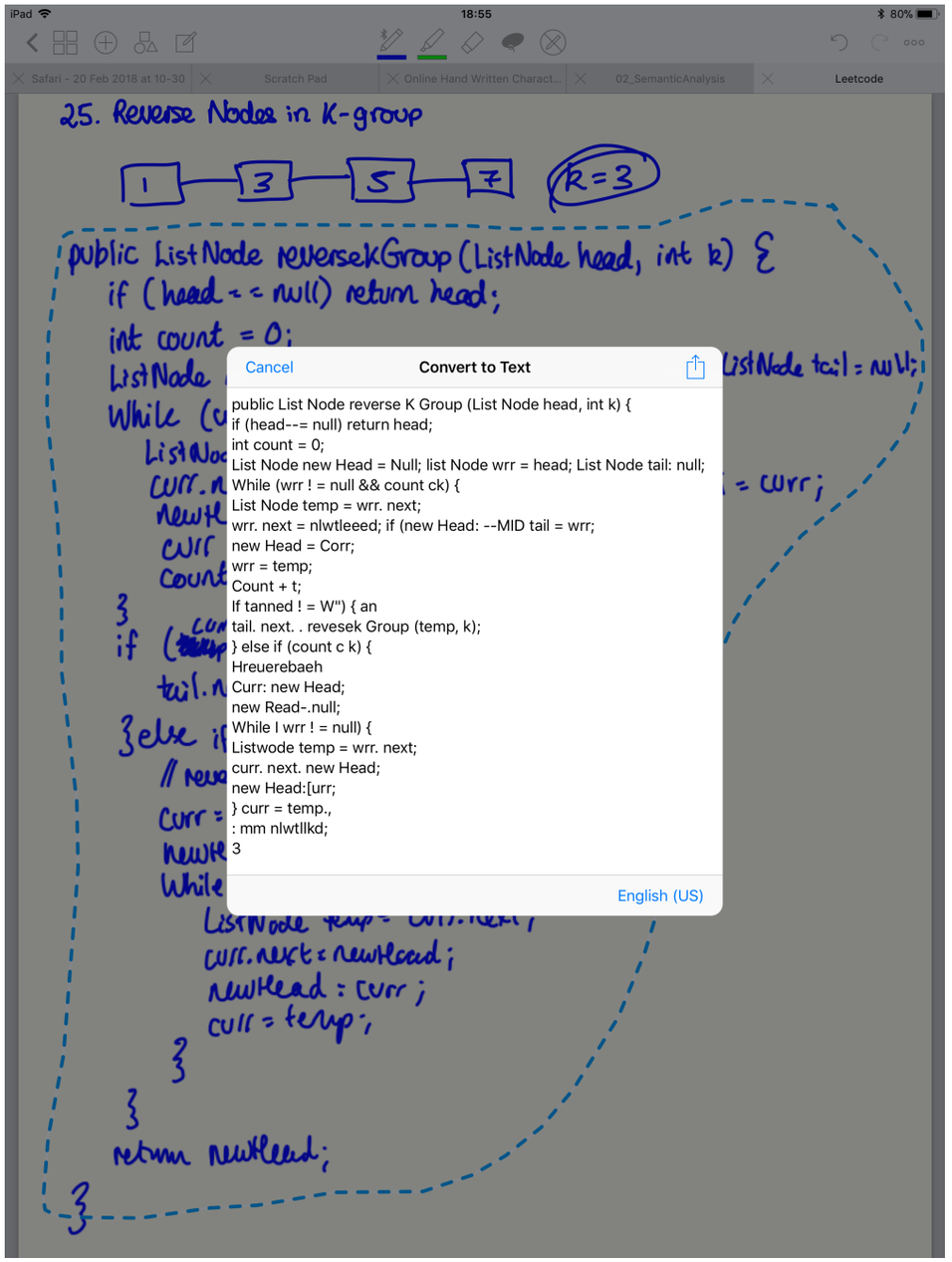
APPROACHES

- Creating a new handwriting recognition system:
  - Hidden Markov Models
  - Convolutional Neural Networks
- Augmenting Existing Systems:
  - Tesseract OCR Engine (offline)
  - MyScript Engine (online)

ONE APPROACH...

- Handwrite on tablet and convert to text.





AUGMENTING TESSERACT (OCR) ENGINE

- Iris - recognise handwritten Ruby code.
  - Tesseract has an API to recognise more fonts.
  - List for frequent words (added keywords of Ruby language)
  - Domain Specific Language created for recognising symbols e.g. ‘plus’, ‘minus’
- Codeable - recognised handwritten C code.
  - Targeted improving pre/post processing.
  - Fix misspellings using edit distance on common variable names.
  - Fix missing delimiters using regular expressions.
  - Always

AUGMENTING MYSCRIPT

- Zhi and Metoyer developed a system to recognised Python.
- Developed the following pipeline:
  - Statement classification (look at first word)
  - Statement parsing
    - Token processing (tracking non keywords)
    - Statement concatenation
- For 45 handwritten code samples:
  - Original: WER 31.31% and CER 9.25%
  - Pipeline: WER 8.6% and CER 3.6%

IMPLEMENTATION DETAILS

- Forked ‘starter’ project from MyScript

EXECUTION – LOCAL SERVER

- Intended for answering problems from a textbook.
- Place code in an empty class or replace a placeholder.
- Create an SSH channel, create code file locally and transfer via SFTP.
- Compile and Execute on server (pipe stderr to stdout)
- Return output to app.

LEETCODE INTEGRATION

- LeetCode website provides access to
  - a number of questions and solutions.
  - Feedback on time and space complexity.
- Does not have an API to access questions / submit solutions.
  - Scrape question information using Python with Selenium.
  - Submit via HTTP
    - First login, then send code in body of request.
    - Set Cross Site Request Forgery (CSRF) token and HTTP headers.

SAVING

- Digital Ink saved in .IINK file
- Word object (contains recognition fixes) saved in .JSON file
- Loading a file will load both the IINK and JSON files
- File Settings stored in UserDefaults
- Support for renaming files

PERSISTENCE OF RECOGNITION FIXES BETWEEN CONVERSIONS

- Convert -> Add/Delete Code
- Gestures to join / break lines
- Compares bounding box labels, width and height values.

SUPPORT FOR MULTIPLE PROGRAMMING LANGUAGES

- Custom Code: Provide commands to compile and execute.
- LeetCode: Change programming language in HTTP request
- ANTLR: Grammar file required
  - JavaScript weakly typed so less checks var i = 0
  - Disable irrelevant stages of pipeline, or write code that works with pipeline. e.g. Swift brackets are optional.
  - Current Indentation issue (no support for Python)