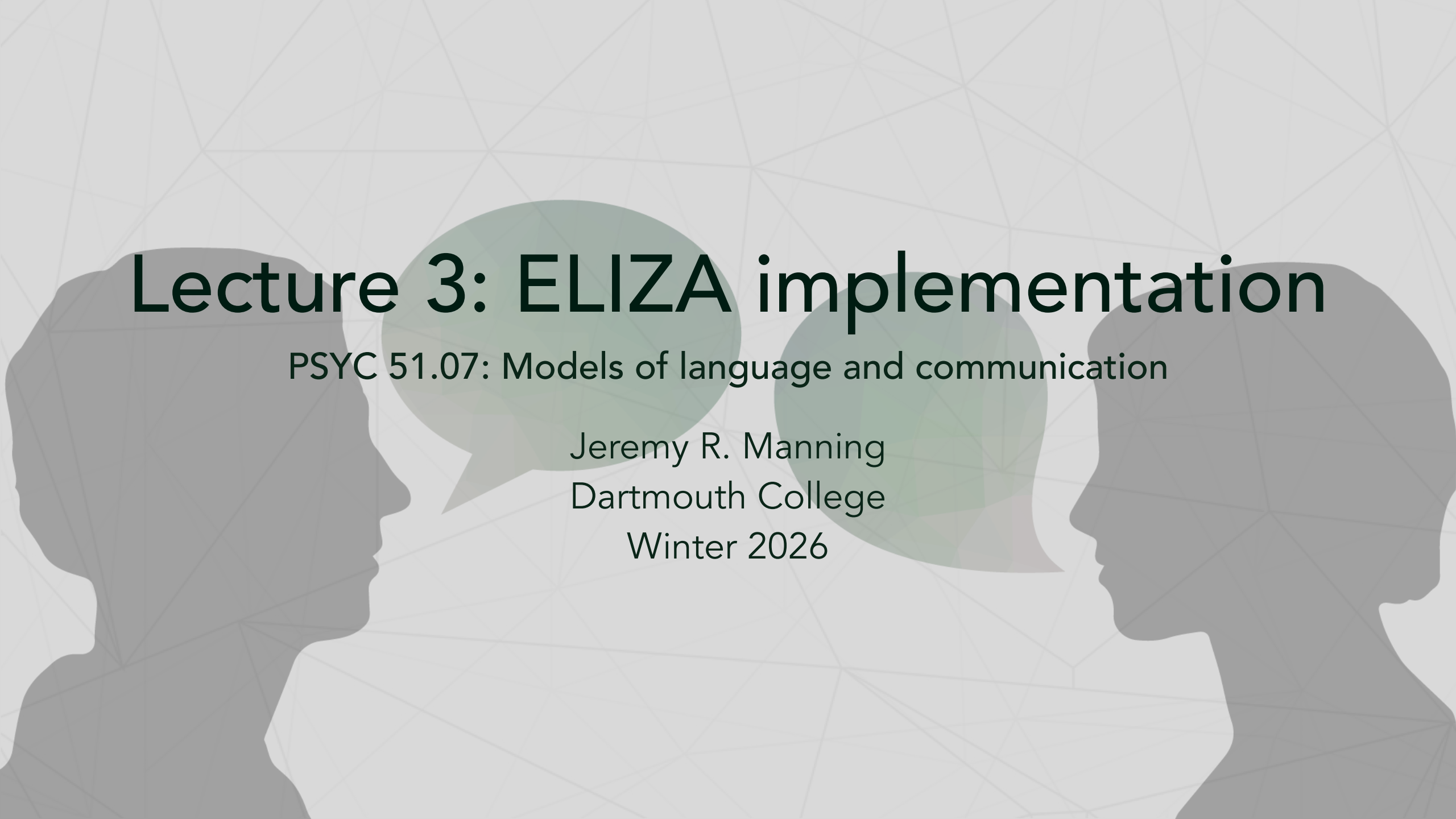


Lecture 3: ELIZA implementation

PSYC 51.07: Models of language and communication

Jeremy R. Manning
Dartmouth College
Winter 2026

Today's agenda

What we'll cover

1. **Algorithm:** how ELIZA actually works, broken down step-by-step
2. **Demo:** playing with a reference implementation to see the algorithm in action
3. **Assignment 1:** tips and tricks for getting started

Goal

Leave today knowing how to build your own implementation of ELIZA.

The ELIZA algorithm

The complete ELIZA processing pipeline

Key insight

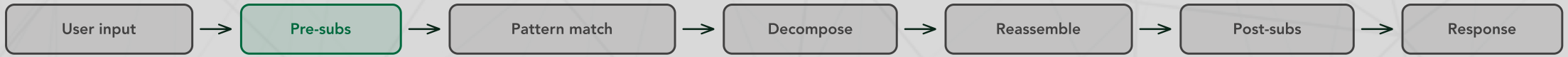
Each step is simple string manipulation. The illusion of "intelligence" comes from three things:

- The *user's* text, which provides structured content
- Hand-crafted patterns and templates that guide the response
- Hard-to-shake human tendencies to anthropomorphize

Under the hood

ELIZA's complete instruction set comprises ~200 simple rules.

Pre-substitutions: normalize input before pattern matching



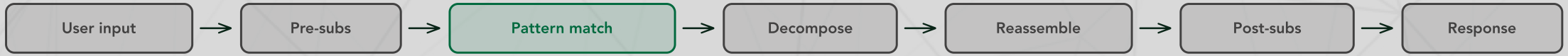
What's the point?

Normalizing common variations helps patterns match more reliably, allowing fewer rules to effectively cover more inputs. This step also handles common misspellings, simplifies phrasing, and helps to set the right tone in the final response.

Example pre-substitutions

- "dont" -> "don't"
- "cant" -> "can't"
- "recollect" -> "remember"
- "how" -> "what"
- "machine" -> "computer"

Pattern matching: detect keywords, synonyms, and patterns in the input



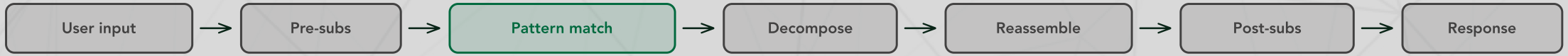
What's the point?

We need to identify which rule to apply based on the user's input. ELIZA uses basic regular expressions to define patterns that can capture important keywords and structure in the input text.

Types of patterns

- **Keywords:** look for specific words (e.g., "mother", "father")
- **Synonyms:** equate different words with similar meanings
- **Sequences:** identify when words appear in a certain order

Pattern matching: detect **keywords**, synonyms, and patterns in the input



How does it work?

ELIZA uses a ranked list of *keywords* to check for. Each keyword is associated with one or more *patterns* based on regular expressions.

Example keywords and ranks

- computer (rank 50)
- dreamed (rank 4)
- everyone (rank 2)

  ...
- xnone (rank 0): fallback when no other patterns match

Pattern matching: detect keywords, **synonyms**, and patterns in the input

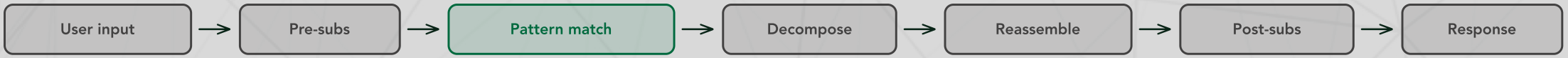
How does it work?

Some keywords have *synonyms*— alternative words with similar meanings. ELIZA treats these as equivalent when matching patterns.

Example synonyms

- belief feel think believe wish
- family mother mom father dad sister brother wife children child
- desire want need
- sad unhappy depressed sick
- happy elated glad better

Pattern matching: detect keywords, synonyms, and **patterns** in the input



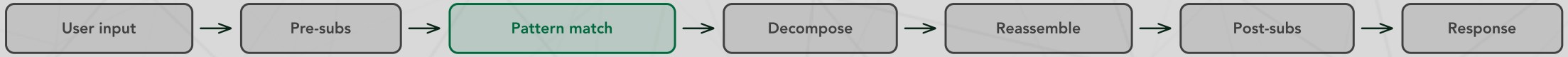
How does it work?

Patterns are defined using simple regular expressions. They can include specific words or wildcards (*). Synonyms are denoted by "@".

Example patterns

keyword	patterns (selected)	matching text
am	* am *; *	I am really excited to build a chatbot
are	* are you *; * are *	Why are you asking me that?
i	* i @desire *; * i am * @happy *; ...	I am so glad to hear that!
yes	*	Yes, I guess that's true but I never really thought about it before.

Pattern matching: memory



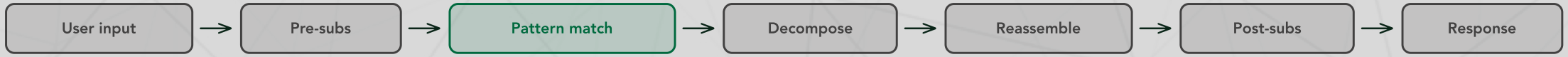
How does it work?

When patterns start with "$", ELIZA saves the matched input to a buffer for later use. When the `xnone` keyword is triggered, ELIZA can pull from this buffer to generate a response to a *previous* input.

Example patterns

In the original rule set, there is only one memory pattern: `$ * my *` (e.g., "I have been thinking a lot about my family"). If no other patterns match later, ELIZA can respond with "Did you come to me to talk about your family?" (or other `* my *` responses) using the saved input.

Pattern matching: rankings



How does it work?

Text is matched against patterns in decreasing order of keyword *rank*. The first pattern that matches is selected for further processing.

Example patterns

Since "computer" has a rank of 50, whereas "dreamed" has a rank of 4, the text "I dreamed about my computer" would be tested against patterns associated with "computer" first. However, if no patterns for higher-ranked keywords match, ELIZA continues down the list until a match is found. The "xnone" keyword (rank 0) provides a fallback pattern (*) when no other patterns match.

Decomposition: extract relevant parts of the input

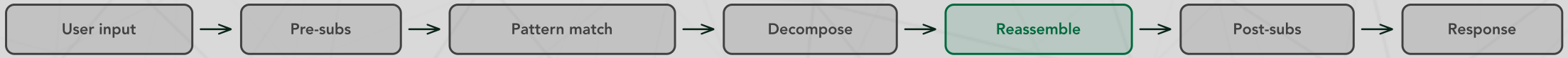
How does it work?

Once text is matched to a pattern, ELIZA breaks down the input into *capture groups* that can be used to craft a response. Each synonym and wildcard (*) in the pattern corresponds to a capture group. Capture groups are numbered in order of appearance, starting from 1.

Example decomposition

Pattern	Input	Captured groups
* i am * @sad	"I am feeling unhappy today"	1: "", 2: "feeling", 3: "unhappy", 4: "today"
* my * @family *	"My mother and father are kind"	1: "", 2: "", 3: "mother", 4: "and father are kind"
*	"I am worried about my dream"	1: "I am worried about my dream"

Reassembly: generate a response using templates



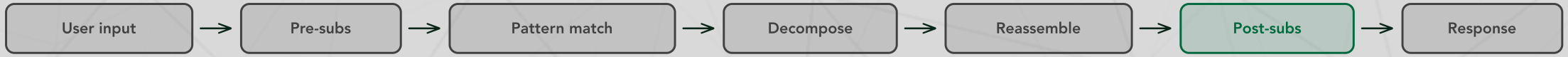
How does it work?

ELIZA's responses "reflect back" parts of the user's input using *reassembly templates*. The templates define how to construct a response— often using capture groups from the decomposition step.

Example reassembly

Input	Captured groups	Reassembly template	Result
"I can't trust people"	1: "", 2: "can't", 3: "trust people"	"Perhaps you could (3) now."	"Perhaps you could trust people now."
"I don't remember why I said that"	1: "", 2: "remember why I said that"	"Do you wish to be able to (2)?"	"Do you wish to be able to remember why you said that?"
"I suppose it wasn't that long ago"	1: "I suppose it wasn't that long ago"	"Do you say (1) for some special reason?"	"Do you say I suppose it wasn't that long ago for some special reason?"

Post-substitutions: reflect first and second person



How does it work?

To ensure the response sounds natural, ELIZA applies *post-substitutions* to adjust pronouns and verb forms. This step flips first-person references to second-person and vice versa. It works similarly to pre-substitutions; it's a simple string replacement based on a predefined dictionary.

Example post-substitutions

- "am" -> "are"
- "your" -> "my"
- "me" -> "you"
- "myself" -> "yourself"
- "i'm" -> "you are"

It's....*demo time*!

With this complete pipeline in mind, let's revisit our ELIZA demo to see how it all fits together. Use the "Rule Breakdown" tab to trace each step of the pipeline for different inputs.

Try it out...

Take a look at the complete list of rules, or view them in the "Live Rule Editor" tab of the demo. Pick out a few patterns and see if you can get ELIZA to pick up on them. Also try to find some edge cases where the pipeline breaks down! For a special challenge, try creating your own rules to see if you can "patch up" the edge cases you find.

How should you approach Assignment 1?

Tips and tricks

1. First, make sure you fully understand the ELIZA algorithm we covered today. Review the slides and/or demo as needed.
2. Use *vibe coding* to accelerate your development. Some strategies are in the Assignment 1 instructions.
3. You **must** carefully test your implementation. Write functions for each step of the pipeline, and test them one-at-a-time. Then integrate them.
4. Consider edge cases and failure modes as you test. What happens if the user input is empty? Or very long? Or contains special characters? What if new rules were added— would your code still work?
5. *Chat with your implementation* by setting up a simple conversation loop. Make sure it behaves as expected; look out for strange or nonsensical responses. Compare to responses generated by the demo implementation.

Debugging tips

Use print statements

```
def find_match(text, rules):
    for rule in rules:
        match = re.search(rule["pattern"], text)
        if match:
            print(f"Matched: {rule['pattern']}")
            print(f"Groups: {match.groups()}")
            return rule, match
    print("No match found!")
    return None, None
```

Print what patterns match and why. Remove prints when done.

Assignment 1 overview

What you'll build

A complete ELIZA implementation that:

- Reads rules from `instructions.txt`
- Handles pre/post substitutions
- Matches patterns and generates responses
- Maintains a conversation loop

Due date

Friday, January 16 (end of Week 2)

Key takeaways

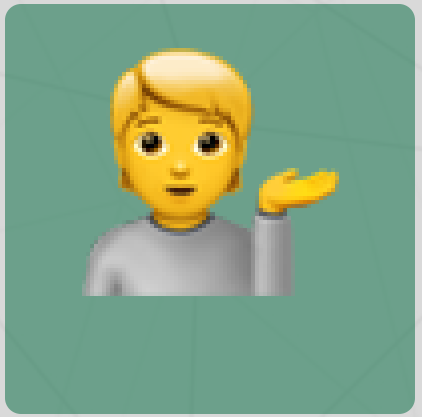
1. **ELIZA is simple:** Just string manipulation and pattern matching
2. **Pre/post subs:** Normalize input, fix output pronouns
3. **Pattern priority:** Check specific patterns before general ones
4. **Vibe coding:** Use AI tools to accelerate development
5. **Test incrementally:** Build and test piece by piece

> **Remember**
>
> The "magic" isn't in the algorithm. It's in how humans interpret the output!

Questions? Want to chat more?

Email me

Join our Discord



Come to office hours

Get help

Start Assignment 1 early. Ask questions, talk to each other, and come to office hours if you get stuck!