

# Workshop: Building a Skinner Teaching Machine in Flask

COMP 395 – AI and Learning Technologies

## Lab 3 Worksheet

# Workshop Overview

**What you'll build:** A web-based teaching machine that implements Skinner's programmed instruction principles.

## Core features:

- Present instructional **frames** one at a time
- Require **active student responses**
- Provide **immediate feedback**
- Allow **self-paced** progression

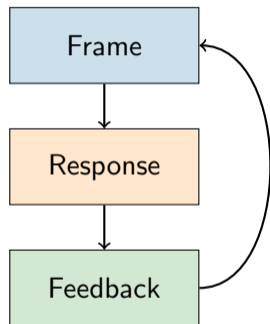
## You will design:

- Your own frame data structure (Python dict)
- A topic of your choice (5–10 frames minimum)
- The Flask routes and HTML templates

# Quick Refresher: Skinner's Teaching Machine

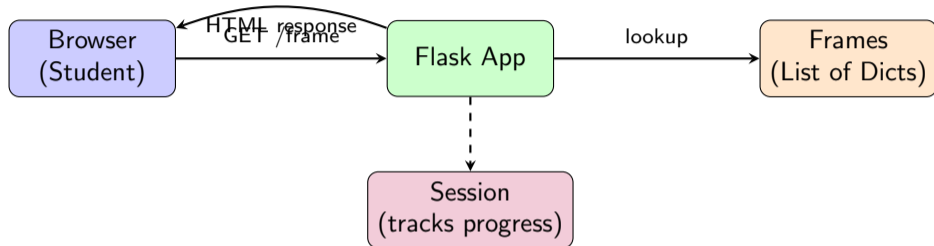
## Key Principles (1958):

- ① **Small steps** – break content into tiny “frames”
- ② **Active responding** – learner must produce an answer
- ③ **Immediate feedback** – instant confirmation
- ④ **Self-pacing** – learner controls speed
- ⑤ **Low error rate** – frames designed for success



*The frame-response-feedback loop*

# System Architecture



## Data flow:

- 1 Student requests current frame → Flask checks session for frame index
- 2 Flask retrieves frame from list → renders HTML template
- 3 Student submits answer → Flask compares, gives feedback, advances

# Part 1: Design Your Frame Structure

## Theory Connection

Skinner emphasized that frames should be small enough that students succeed  $\sim 95\%$  of the time. Each frame teaches *one* small concept.

A frame needs (at minimum):

- **Content/prompt** – what the student sees
- **Expected answer** – what counts as correct
- **Feedback** – what to show after responding

## Your Task

Decide: What additional fields might be useful? (hints? difficulty? topic tags?)

# Example Frame Structures

## Option A: Minimal

```
frame = {  
    prompt : The capital of France is _____. ,  
    answer : paris  
}
```

## Option B: With feedback

```
frame = {  
    prompt : The capital of France is _____. ,  
    answer : paris ,  
    feedback_correct : Correct! Paris is the capital. ,  
    feedback_incorrect : Not quite. The answer is Paris.  
}
```

## Example Frame Structures (continued)

### Option C: Rich frame with hints and multiple acceptable answers

```
frame = {  
    id : 1,  
    prompt : What operator adds two numbers in Python? ,  
    answers : [ + , plus , addition ], # multiple accepted  
    hint : It's a single character on your keyboard. ,  
    explanation : The + operator performs addition. ,  
    topic : python-basics  
}
```

#### Your Task

Sketch your frame structure in your notebook. What fields will you include? You'll implement this next.

## Part 2: Create Your Frames

Create a file called `frames.py` with your teaching content:

```
# frames.py
# Topic: [YOUR TOPIC HERE]

FRAMES = [
    {
        prompt : Your first frame prompt here... ,
        answer : expected answer ,
        feedback_correct : Great job! ,
        feedback_incorrect : Try again. The answer is...
    },
    {
        prompt : Your second frame... ,
        answer : answer2 ,
        feedback_correct : Excellent! ,
        feedback_incorrect : Not quite...
    },
]
```

# Frame Design Tips

## Theory Connection

Skinner's frames used **constructed responses** (fill-in-the-blank) rather than multiple choice, because producing an answer requires deeper processing than recognition.

## Good frame design:

- Start with easier frames, gradually increase difficulty
- Each frame should build on previous ones
- Use blanks strategically: ‘‘In Python, we use \_\_\_\_\_ to define a function.’’
- Keep prompts concise and unambiguous
- Normalize answers (lowercase, strip whitespace) for comparison

## Your Task

Write at least 5 frames for a topic you know well. Consider: What's the logical sequence?

## Part 3: Flask App Structure

Create app.py with this skeleton:

```
from flask import Flask, render_template, request, session, redirect
    , url_for
from frames import FRAMES

app = Flask(__name__)
app.secret_key = your-secret-key-here # needed for sessions

@app.route( / )
def index():
    # Initialize session, show welcome or first frame
    pass

@app.route( /frame )
def show_frame():
    # Display current frame
    pass
```

# Implementing the Index Route

```
@app.route( / )
def index():
    # Reset progress when starting fresh
    session[ current_frame ] = 0
    session[ score ] = 0
    return render_template( index.html , total_frames=len(FRAMES))
```

## What's happening:

- session is a Flask dict that persists across requests
- We store the current frame index (starts at 0)
- We track score for optional gamification
- We render a welcome template

# Implementing the Frame Display Route

```
@app.route( /frame )
def show_frame():
    frame_idx = session.get( current_frame , 0)

    # Check if we've completed all frames
    if frame_idx >= len(FRAMES):
        return redirect(url_for( complete ))

    frame = FRAMES[frame_idx]
    return render_template(
        frame.html ,
        frame=frame ,
        frame_num=frame_idx + 1 ,
        total=len(FRAMES)
    )
```

# Implementing Answer Submission

```
@app.route( /submit , methods=[ POST ])
def submit_answer():
    frame_idx = session.get( current_frame , 0)
    frame = FRAMES[frame_idx]

    user_answer = request.form.get( answer , ).strip().lower()
    correct_answer = frame[ answer ].strip().lower()

    is_correct = (user_answer == correct_answer)

    if is_correct:
        session[ score ] = session.get( score , 0) + 1
        session[ current_frame ] = frame_idx + 1

    feedback_key = feedback_correct if is_correct else
        feedback_incorrect
    feedback = frame.get(feedback_key, Correct! if is_correct else
```

# Implementing the Completion Route

```
@app.route( /complete )
def complete():
    score = session.get( score , 0)
    total = len(FRAMES)
    percentage = round((score / total) * 100) if total > 0 else 0

    return render_template( complete.html ,
        score=score,
        total=total,
        percentage=percentage
    )
```

## Theory Connection

Skinner's machines provided cumulative feedback. Students could see their progress—a form of **continuous reinforcement** that maintains motivation.

## Part 4: HTML Templates

Create a `templates/` folder with these files:

File	Purpose
<code>base.html</code>	Shared layout (header, CSS)
<code>index.html</code>	Welcome page, “Start” button
<code>frame.html</code>	Displays prompt, answer input
<code>feedback.html</code>	Shows correct/incorrect, “Next” button
<code>complete.html</code>	Final score, “Restart” option

### Project structure:

```
your-project/  
  app.py  
  frames.py  
  templates/  
    base.html  
    index.html
```

# Template: base.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Teaching Machine</title>
  <style>
    body { font-family: Arial, sans-serif; max-width: 600px;
           margin: 50px auto; padding: 20px; }
    .prompt { font-size: 1.3em; margin: 20px 0; }
    input[type= text ] { font-size: 1.1em; padding: 10px; width:
      100%; }
    button { font-size: 1.1em; padding: 10px 20px; margin-top:
      15px;
             cursor: pointer; }
    .correct { color: green; font-weight: bold; }
    .incorrect { color: red; font-weight: bold; }
    .progress { color: gray; font-size: 0.9em; }
  </style>
</head>
```

# Template: index.html

```
{% extends base.html %}

{% block content %}
<h1>Welcome to the Teaching Machine</h1>

<p>This lesson contains <strong>{{ total_frames }}</strong> frames.</p>

<p>Instructions:</p>
<ul>
  <li>Read each prompt carefully</li>
  <li>Type your answer in the box</li>
  <li>Get immediate feedback</li>
  <li>Progress at your own pace</li>
</ul>

<a href= {{ url_for('show_frame') }} >
```

# Template: frame.html

```
{% extends    base.html    %}

{% block content %}
<p class= progress >Frame {{ frame_num }} of {{ total }}</p>

<div class= prompt >
    {{ frame.prompt }}
</div>

<form action= {{ url_for('submit_answer') }} method= POST >
    <input type= text    name= answer    placeholder= Type your answer
        ...
        autofocus required>
    <br>
    <button type= submit >Submit</button>
</form>
{% endblock %}
```

# Template: feedback.html

```
{% extends base.html %}

{% block content %}
<h2 class= {{ 'correct' if is_correct else 'incorrect' }} >
    {{ Correct! if is_correct else Not quite... }}
</h2>

<p>{{ feedback }}</p>

{% if not is_correct %}
<p><em>The correct answer was: <strong>{{ frame.answer }}</strong></em></p>
{% endif %}

<a href= {{ url_for('show_frame') }} >
    <button>{{ Next Frame if is_correct else Try Next Frame }}</button>
</a>
```

# Template: complete.html

```
{% extends base.html %}

{% block content %}
<h1>Lesson Complete!</h1>

<p>You scored <strong>{{ score }}</strong> out of <strong>{{ total
    }}</strong>
    ({{ percentage }}%)</p>

{% if percentage == 100 %}
    <p>Perfect score! Excellent work!</p>
{% elif percentage >= 80 %}
    <p>Great job! You've mastered most of the material.</p>
{% else %}
    <p>Keep practicing! Consider reviewing and trying again.</p>
{% endif %}
```

## Part 5: Running Your Teaching Machine

### 1. Make sure your environment is active:

```
conda activate flask-env
```

### 2. Run the Flask app:

```
python app.py
```

### 3. Open your browser:

```
http://127.0.0.1:5000/
```

### 4. Test the full flow:

- Start → Answer frames → Get feedback → Complete
- Test both correct and incorrect answers
- Verify score tracking works

# Extension Ideas (Optional Challenges)

## Enhance your teaching machine:

- 1 **Hints system** – Add a “Show Hint” button that reveals `frame["hint"]`
- 2 **Multiple acceptable answers** – Change `answer` to a list; check if user's answer is in the list
- 3 **Retry on incorrect** – Don't advance until the student gets it right (closer to Skinner's original design)
- 4 **Branching frames** – If incorrect, insert a remedial frame before continuing
- 5 **Timing data** – Record how long each frame takes; store in session
- 6 **Persistent storage** – Save results to a JSON file or database

### Theory Connection

Branching frames move toward Crowder's “intrinsic programming”—a different theory! Consider: what are the tradeoffs?

# Reflection Questions

After building your teaching machine, consider:

- ➊ How did designing frames change your understanding of the content you chose?
- ➋ What was difficult about breaking knowledge into small steps?
- ➌ Skinner claimed teaching machines could replace teachers for factual content. Do you agree? What can't this approach do?
- ➍ How might you combine this approach with modern AI (e.g., generating feedback dynamically)?
- ➎ What would Papert (constructionism) say about this approach to learning?

# Completion Checklist

Before you finish, verify:

- ☐ `frames.py` contains at least 5 well-designed frames
- ☐ Frame structure is consistent (all frames have same keys)
- ☐ `app.py` has all routes implemented
- ☐ All 5 templates exist and render correctly
- ☐ App starts without errors
- ☐ Can complete full flow: start → frames → feedback → complete
- ☐ Score tracks correctly
- ☐ Both correct and incorrect answers handled properly

## Bonus:

- ☐ Implemented at least one extension feature
- ☐ Wrote thoughtful reflection responses

## Reference: Complete app.py

```
from flask import Flask, render_template, request, session, redirect, url_for
from frames import FRAMES

app = Flask(__name__)
app.secret_key = change-this-to-something-secret

@app.route( / )
def index():
    session[ current_frame ] = 0
    session[ score ] = 0
    return render_template( index.html , total_frames=len(FRAMES))

@app.route( /frame )
def show_frame():
    frame_idx = session.get( current_frame , 0)
    if frame_idx >= len(FRAMES):
        return redirect(url_for( complete ))
    frame = FRAMES[frame_idx]
    return render_template( frame.html , frame=frame,
                           frame_num=frame_idx + 1, total=len(FRAMES))
```

## Reference: Complete app.py (continued)

```
@app.route( /submit , methods=[ POST ])
def submit_answer():
    frame_idx = session.get( current_frame , 0)
    frame = FRAMES[frame_idx]
    user_answer = request.form.get( answer , ).strip().lower()
    correct_answer = frame[ answer ].strip().lower()
    is_correct = (user_answer == correct_answer)
    if is_correct:
        session[ score ] = session.get( score , 0) + 1
        session[ current_frame ] = frame_idx + 1
    feedback_key = feedback_correct if is_correct else feedback_incorrect
    feedback = frame.get(feedback_key, Correct! if is_correct else Incorrect. )
    return render_template( feedback.html , is_correct=is_correct,
                           feedback=feedback, frame=frame)

@app.route( /complete )
def complete():
    score = session.get( score , 0)
    total = len(FRAMES)
    percentage = round((score / total) * 100) if total > 0 else 0
    return render_template( complete.html , score=score, total=total,
                           percentage=percentage)
```