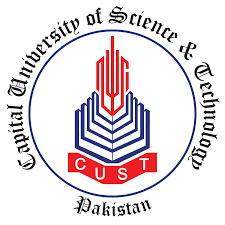
**“SmartRule: Dynamic Evaluation of User-Defined Policies”**

**BCS223205 – Khawaja Saad Ayaz**

**Section: 2**



**Submitted to: Ms. Mahin Malik**

**Department of Computer Science**

**Capital University of Science & Technology, Islamabad**

**Table of Contents**

[**CHAPTER 1** 3](#_Toc201695569)

[**1.1 Introduction** 3](#_Toc201695570)

[**CHAPTER 2** 4](#_Toc201695571)

[**2.1 Problem Statement** 4](#_Toc201695572)

[**CHAPTER 3** 5](#_Toc201695573)

[**Methodology** 5](#_Toc201695574)

[**3.1 Development Approach** 5](#_Toc201695575)

[**3.2 Application Workflow** 6](#_Toc201695576)

[**3.3 Tools and Libraries Used** 7](#_Toc201695577)

[**3.4 Design Considerations** 7](#_Toc201695578)

[**3.5 Theory of Programming Languages Concepts** 8](#_Toc201695579)

[**3.6 Flowchart** 10](#_Toc201695580)

[**CHAPTER 4** 11](#_Toc201695581)

[**Conclusion** 11](#_Toc201695582)

[**CHAPTER 5** 12](#_Toc201695583)

[**Code** 12](#_Toc201695584)

[**Output** 19](#_Toc201695585)

[**Main Page** 19](#_Toc201695586)

[**Add Rule** 19](#_Toc201695587)

[**Show Rule** 20](#_Toc201695588)

[**Edit Rule** 21](#_Toc201695589)

[**Delete Rule** 22](#_Toc201695590)

[**Upload Rules** 24](#_Toc201695591)

[**Evaluation** 25](#_Toc201695592)

[**Parse Tree of Rule** 25](#_Toc201695593)

# **CHAPTER 1**

## **1.1 Introduction**

In modern-day software systems, many business applications rely heavily on conditional logic for instance, assigning grades to students based on marks, computing bonus based on sales, or classifying user profiles based on age or category. Traditionally, such rules are embedded directly into source code by developers. This makes the system inflexible and difficult to update by non-technical users.

To address this gap, I have designed and developed a project titled **SmartRule: A Dynamic Rule Evaluation Engine**. It allows users to define rules in a simple format such as:

**if marks > 85 then grade = 'A'**

The system interprets and evaluates these rules against live input data. A key strength of this tool is that it brings together both **theory** (parsing, grammars, ambiguity, semantics) and **application** (real-world decision logic).

SmartRule was built using Python and includes a full GUI, rule editor, file upload support, and parse tree visualization.

# **CHAPTER 2**

## **2.1 Problem Statement**

In organizations and academic institutions, business logic is often subject to change. For instance:

* Grading policies may vary by department.
* Discount rules may change during sales.
* Bonus criteria may differ across teams.

When these rules are hardcoded, every update requires developer intervention. Moreover, non-programmers have no way to define or test their own rules dynamically.

**The core problem this project addresses is:**  
*How can users dynamically define and evaluate conditional rules without writing full source code, while ensuring correctness and structure?*

SmartRule solves this by introducing a DSL (domain-specific language) to define rules, a parser to check their validity, and a dynamic evaluator to compute outputs. The entire process reflects key topics from the Theory of Programming Languages.

# **CHAPTER 3**

## **Methodology**

### **3.1 Development Approach**

SmartRule was developed using Python with an object-oriented design. Two core classes were created:

* **Rule** – encapsulates a single rule with a condition and an action.
* **SmartRuleEngine** – manages the collection of rules, parsing, evaluation, and output.

The frontend GUI was created using the **tkinter** library. This allows the user to:

* Add new rules.
* View, edit, and delete existing rules.
* Upload and load rules from files.
* Evaluate rules with input data.
* Visualize the structure of rules using parse trees.

All user input is processed in real-time, with dynamic behavior handled using Python’s **eval()** and **exec()** functions.

### **3.2 Application Workflow**

#### **3.2.1 Rule Entry:**

The user defines rules in the format *if <condition> then <action>* using a text input field in the GUI.

#### **3.2.2 Parsing:**

Each rule is parsed using a regular expression to separate the condition and action components. The format strictly follows a **context-free grammar**, ensuring structure.

#### **3.2.3 Rule Management:**

The user can view all defined rules, edit them, delete any rule, or upload a rule file (e.g., ***rules.txt***). This supports dynamic rule handling.

#### **3.2.4 Input Entry:**

The user enters runtime input data (e.g., ***marks = 88***, ***sales = 120000***). These variables are passed into the evaluation engine.

#### **3.2.5 Evaluation:**

The engine loops through each rule. If the condition evaluates to True, the corresponding action is executed using ***exec()***, modifying the input dictionary.

#### **3.2.6 Output:**

All applied rules and the final data are displayed in the output sections. Parse tree visualization is shown using the ***graphviz*** library.

### **3.3 Tools and Libraries Used**

|  |  |
| --- | --- |
| Tool / Library | Purpose |
| Python 3.x | Base language |
| tkinter | GUI interface |
| re | Regular expressions for parsing |
| graphviz | Parse tree visualization |
| filedialog / os | File selection and I/O |
| eval / exec | Dynamic evaluation of rule logic |

Table 1: Tools &Libraries used

### **3.4 Design Considerations**

* Rules must follow a strict yet simple format to support parsing.
* GUI must be intuitive for both technical and non-technical users.
* System should support saving and loading rule sets via .txt files.
* Users should be able to visualize how rules are structured (parse tree).
* Safety must be ensured during dynamic execution (e.g., invalid inputs, unsafe commands).

### **3.5 Theory of Programming Languages Concepts**

#### **3.5.1 Context-Free Grammar (CFG):**

Rules are defined in the form:

***rule → if <condition> then <action>***

This structure forms a simple CFG, where ***<condition>*** and ***<action>*** are Python expressions.

#### **3.5.2 Parse Trees:**

Rules are split into two logical nodes under a root if. The project visualizes this using graphviz, generating trees like:

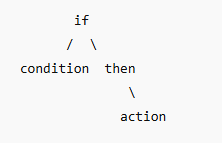


Figure 1: General Parse Tree

#### **3.5.3 Ambiguity Detection:**

While not auto-detected, the system allows manual identification of overlapping or conflicting rules, which may behave ambiguously depending on input.

#### **3.5.4 Static Semantics:**

Rule syntax is checked during parsing to ensure correct format. Invalid rules (e.g., missing ***then***) are rejected before runtime.

#### **3.5.5 Dynamic Semantics:**

During rule evaluation, conditions are tested against live user input and actions are executed dynamically. This simulates dynamic semantics behavior in interpreters.

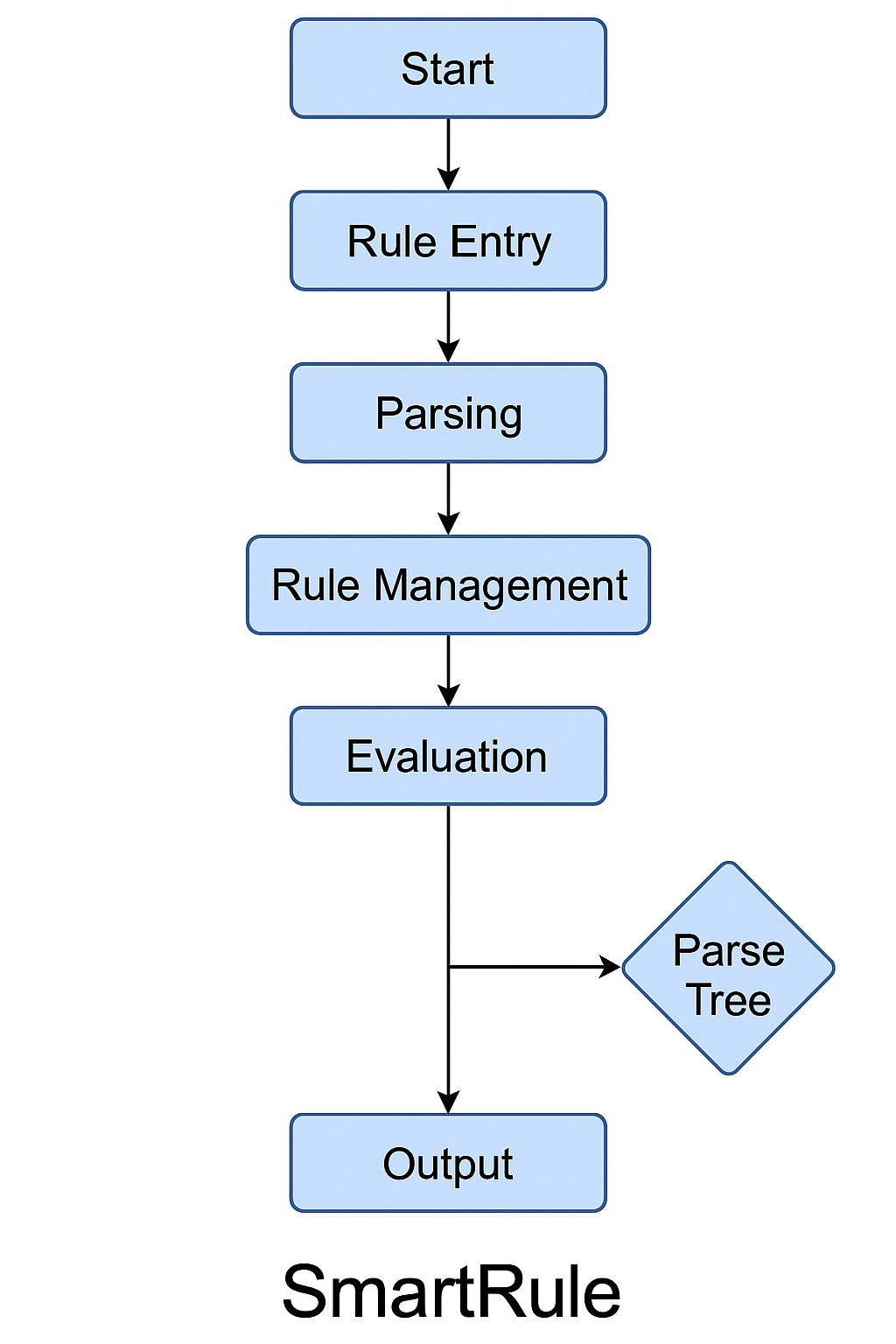
#### **3.5.6 Data Abstraction:**

Rules, conditions, actions, and input data are abstracted using classes and dictionaries. The ***Rule*** class encapsulates logic, and the ***SmartRuleEngine*** abstracts rule management.

#### **3.5.7 Subrange and Enumeration:**

Conditions like ***0 <= marks <= 100*** or ***grade in {'A','B','C'}*** show how subranges and enumerations are handled during rule definition and evaluation.

### **3.6 Flowchart**



# **CHAPTER 4**

## **Conclusion**

SmartRule is a successful demonstration of how **theoretical programming language concepts** can be applied to **real-world software problems**. It is flexible, user-friendly, and supports dynamic behavior without compromising structure or safety.

Through this project, I implemented and practically understood:

* Context-free grammars for rule definition
* Parse tree construction and visualization
* Dynamic and static semantics in a real interpreter
* Data abstraction using OOP
* Real-world usage of subranges and enumerations

The final application allows rules to be added, modified, visualized, and executed in a structured and academic way — fulfilling the learning objectives of this course.

# **CHAPTER 5**

## **Code**

import tkinter as tk

from tkinter import messagebox, scrolledtext, Toplevel, filedialog

from graphviz import Digraph

# =============================

# Data Abstraction

# =============================

class Rule:

    def \_\_init\_\_(self, condition, action):

        self.condition = condition.strip()

        self.action = action.strip()

    # =============================

    # Dynamic Programming

    # =============================

    def evaluate(self, data):

        try:

            if eval(self.condition, {}, data):

                exec(self.action, {}, data)

                return True

            return False

        except Exception as e:

            print(f"Error: {e}")

            return False

# =============================

# CFG + Parse Tree

# =============================

class SmartRuleEngine:

    def \_\_init\_\_(self):

        self.rules = []

    def add\_rule(self, rule\_str):

        import re

        match = re.match(r"if (.+) then (.+)", rule\_str.strip())

        if match:

            condition, action = match.groups()

            rule = Rule(condition, action)

            self.rules.append(rule)

            return True

        return False

    def get\_rules(self):

        return self.rules

    def evaluate\_rules(self, data):

        applied\_rules = []

        for rule in self.rules:

            if rule.evaluate(data):

                applied\_rules.append(f"IF {rule.condition} THEN {rule.action}")

        return applied\_rules, data

# =============================

# GUI with Styling + Features

# =============================

class SmartRuleApp:

    def \_\_init\_\_(self, root):

        self.engine = SmartRuleEngine()

        root.title("SmartRule: Rule-Based Evaluator")

        root.geometry("720x650")

        root.configure(bg="#f0f4f8")

        menu = tk.Menu(root)

        root.config(menu=menu)

        file\_menu = tk.Menu(menu, tearoff=0)

        menu.add\_cascade(label="File", menu=file\_menu)

        file\_menu.add\_command(label="Save Rules", command=self.save\_rules)

        file\_menu.add\_command(label="Load Rules", command=self.load\_rules)

        heading = tk.Label(root, text="SmartRule Engine", font=("Helvetica", 20, "bold"), bg="#f0f4f8", fg="#003366")

        heading.pack(pady=10)

        tk.Label(root, text="Enter Rule (if ... then ...):", bg="#f0f4f8", font=("Arial", 12)).pack()

        self.rule\_entry = tk.Entry(root, width=80, font=("Arial", 11))

        self.rule\_entry.pack(pady=5)

        button\_frame = tk.Frame(root, bg="#f0f4f8")

        button\_frame.pack(pady=5)

        tk.Button(button\_frame, text="Add Rule", command=self.add\_rule, width=15, bg="#007acc", fg="white").grid(row=0, column=0, padx=10)

        tk.Button(button\_frame, text="Show Rules", command=self.show\_rules, width=15, bg="#6a1b9a", fg="white").grid(row=0, column=1)

        tk.Button(button\_frame, text="Upload Rules File", command=self.upload\_rules\_file, width=20, bg="#ffa000", fg="black").grid(row=0, column=2, padx=10)

        tk.Label(root, text="Enter Input Data (e.g., marks=85):", bg="#f0f4f8", font=("Arial", 12)).pack(pady=(20, 0))

        self.input\_text = scrolledtext.ScrolledText(root, width=60, height=5, font=("Consolas", 11))

        self.input\_text.pack(pady=5)

        tk.Button(root, text="Run Evaluation", command=self.run\_engine, width=20, bg="green", fg="white", font=("Arial", 11, "bold")).pack(pady=10)

        tk.Label(root, text="Rules Applied:", bg="#f0f4f8", font=("Arial", 12)).pack()

        self.rules\_output = scrolledtext.ScrolledText(root, width=80, height=7, font=("Consolas", 11), bg="#e8f5e9")

        self.rules\_output.pack(pady=5)

        tk.Label(root, text="Final Data Output:", bg="#f0f4f8", font=("Arial", 12)).pack()

        self.final\_data\_output = tk.Text(root, height=5, width=80, font=("Consolas", 11), bg="#fff3e0")

        self.final\_data\_output.pack(pady=5)

    def add\_rule(self):

        rule = self.rule\_entry.get()

        if self.engine.add\_rule(rule):

            messagebox.showinfo("Rule Added", f"Added Rule:\n{rule}")

            self.rule\_entry.delete(0, tk.END)

        else:

            messagebox.showerror("Invalid Rule", "Use format: if <condition> then <action>")

    def show\_rules(self):

        rules\_window = Toplevel()

        rules\_window.title("Saved Rules")

        rules\_window.geometry("550x500")

        rules\_window.configure(bg="#f5f5f5")

        tk.Label(rules\_window, text="All Defined Rules", font=("Arial", 14, "bold"), bg="#f5f5f5").pack(pady=10)

        self.rule\_listbox = tk.Listbox(rules\_window, width=70, height=10, font=("Consolas", 11))

        self.rule\_listbox.pack(padx=10, pady=5)

        self.rules\_in\_popup = self.engine.get\_rules()

        for idx, r in enumerate(self.rules\_in\_popup):

            self.rule\_listbox.insert(tk.END, f"{idx+1}. IF {r.condition} THEN {r.action}")

        tk.Label(rules\_window, text="Edit Selected Rule:", bg="#f5f5f5", font=("Arial", 12)).pack(pady=(10, 0))

        self.edit\_entry = tk.Entry(rules\_window, width=70, font=("Arial", 11))

        self.edit\_entry.pack(pady=5)

        edit\_frame = tk.Frame(rules\_window, bg="#f5f5f5")

        edit\_frame.pack(pady=5)

        tk.Button(edit\_frame, text="Load for Editing", command=self.load\_selected\_rule, width=15, bg="#007acc", fg="white").grid(row=0, column=0, padx=10)

        tk.Button(edit\_frame, text="Update Rule", command=self.update\_rule, width=15, bg="#4caf50", fg="white").grid(row=0, column=1)

        tk.Button(edit\_frame, text="Delete Rule", command=self.delete\_rule, width=15, bg="#e53935", fg="white").grid(row=0, column=2, padx=10)

        tk.Button(rules\_window, text="Show Parse Tree", command=self.show\_parse\_tree, bg="#ff9800", fg="white").pack(pady=10)

    def load\_selected\_rule(self):

        selected = self.rule\_listbox.curselection()

        if not selected:

            messagebox.showwarning("No selection", "Please select a rule to edit.")

            return

        idx = selected[0]

        rule = self.rules\_in\_popup[idx]

        self.edit\_entry.delete(0, tk.END)

        self.edit\_entry.insert(0, f"if {rule.condition} then {rule.action}")

        self.current\_edit\_index = idx

    def update\_rule(self):

        if not hasattr(self, 'current\_edit\_index'):

            messagebox.showwarning("No rule loaded", "Please load a rule first.")

            return

        updated\_rule = self.edit\_entry.get().strip()

        import re

        match = re.match(r"if (.+) then (.+)", updated\_rule)

        if not match:

            messagebox.showerror("Invalid Format", "Use format: if <condition> then <action>")

            return

        condition, action = match.groups()

        new\_rule = Rule(condition, action)

        self.engine.rules[self.current\_edit\_index] = new\_rule

        self.rule\_listbox.delete(0, tk.END)

        for idx, r in enumerate(self.engine.rules):

            self.rule\_listbox.insert(tk.END, f"{idx+1}. IF {r.condition} THEN {r.action}")

        messagebox.showinfo("Rule Updated", "Rule updated successfully.")

    def delete\_rule(self):

        if not hasattr(self, 'current\_edit\_index'):

            messagebox.showwarning("No rule selected", "Please load a rule to delete.")

            return

        confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete this rule?")

        if confirm:

            del self.engine.rules[self.current\_edit\_index]

            self.rule\_listbox.delete(0, tk.END)

            for idx, r in enumerate(self.engine.rules):

                self.rule\_listbox.insert(tk.END, f"{idx+1}. IF {r.condition} THEN {r.action}")

            self.edit\_entry.delete(0, tk.END)

            messagebox.showinfo("Deleted", "Rule deleted successfully.")

    def show\_parse\_tree(self):

        selected = self.rule\_listbox.curselection()

        if not selected:

            messagebox.showwarning("No rule", "Please select a rule.")

            return

        idx = selected[0]

        rule = self.rules\_in\_popup[idx]

        dot = Digraph()

        dot.node('IF', 'if')

        dot.node('COND', rule.condition)

        dot.node('THEN', 'then')

        dot.node('ACT', rule.action)

        dot.edges([('IF', 'COND'), ('IF', 'THEN'), ('THEN', 'ACT')])

        dot.render('parse\_tree', view=True, format='png')

    def save\_rules(self):

        try:

            with open("rules.txt", "w") as f:

                for rule in self.engine.rules:

                    f.write(f"if {rule.condition} then {rule.action}\n")

            messagebox.showinfo("Saved", "Rules saved to rules.txt")

        except Exception as e:

            messagebox.showerror("Error", str(e))

    def load\_rules(self):

        try:

            with open("rules.txt", "r") as f:

                self.engine.rules.clear()

                for line in f:

                    self.engine.add\_rule(line.strip())

            messagebox.showinfo("Loaded", "Rules loaded from rules.txt")

        except Exception as e:

            messagebox.showerror("Error", str(e))

    def upload\_rules\_file(self):

        file\_path = filedialog.askopenfilename(filetypes=[("Text files", "\*.txt")])

        if file\_path:

            try:

                with open(file\_path, "r") as f:

                    self.engine.rules.clear()

                    for line in f:

                        self.engine.add\_rule(line.strip())

                messagebox.showinfo("Uploaded", f"Rules loaded from {file\_path}")

            except Exception as e:

                messagebox.showerror("Error", str(e))

    def run\_engine(self):

        input\_lines = self.input\_text.get("1.0", tk.END).strip().splitlines()

        data = {}

        try:

            for line in input\_lines:

                if '=' in line:

                    key, value = line.split('=')

                    data[key.strip()] = eval(value.strip())

        except:

            messagebox.showerror("Input Error", "Invalid input format. Use key=value")

            return

        applied, final\_data = self.engine.evaluate\_rules(data)

        self.rules\_output.delete("1.0", tk.END)

        if applied:

            for r in applied:

                self.rules\_output.insert(tk.END, f"{r}\n")

        else:

            self.rules\_output.insert(tk.END, "No rules applied.\n")

        self.final\_data\_output.delete("1.0", tk.END)

        self.final\_data\_output.insert(tk.END, str(final\_data))

# =============================

# Main Loop

# =============================

if \_\_name\_\_ == "\_\_main\_\_":

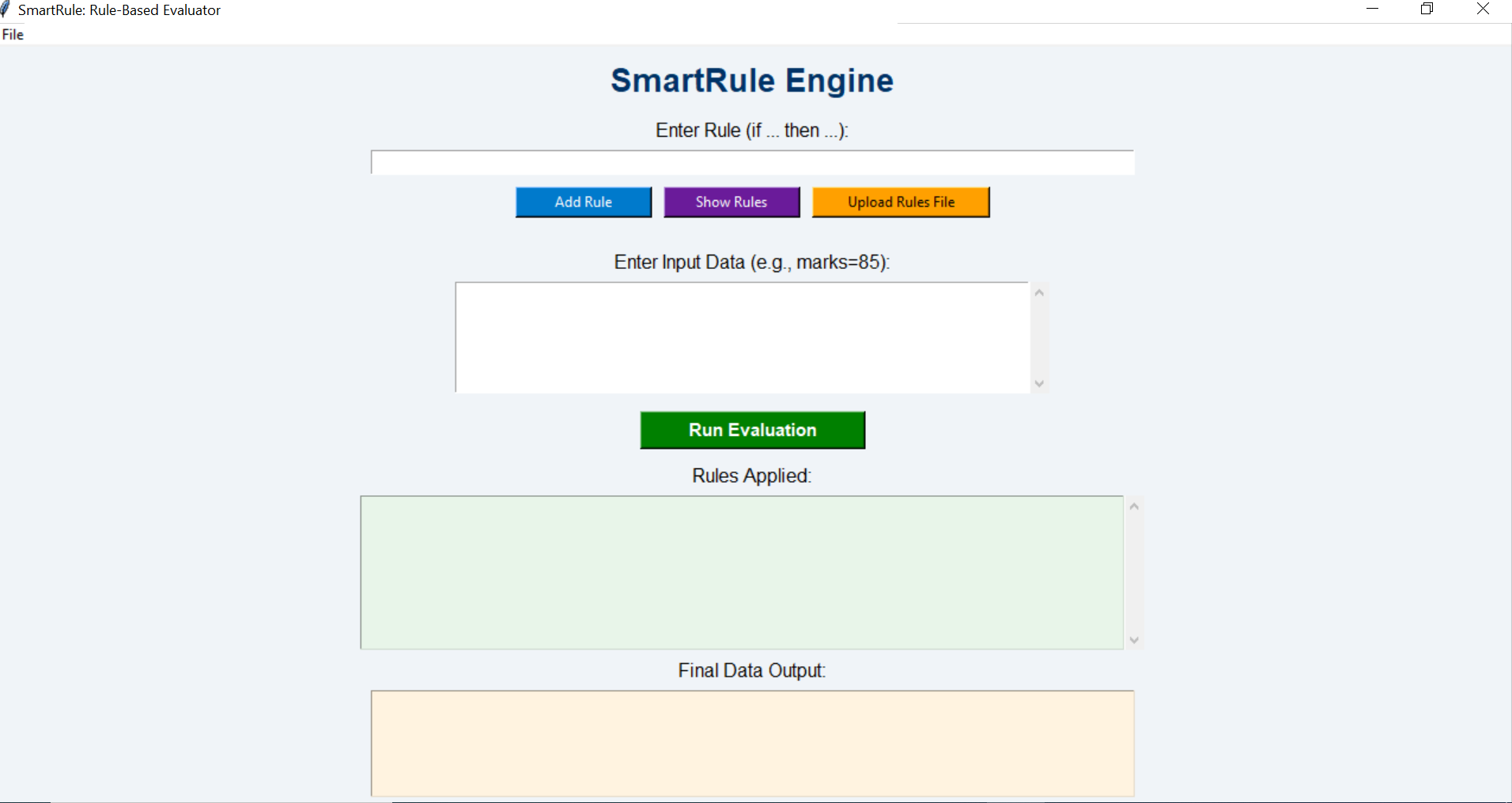
    root = tk.Tk()

    app = SmartRuleApp(root)

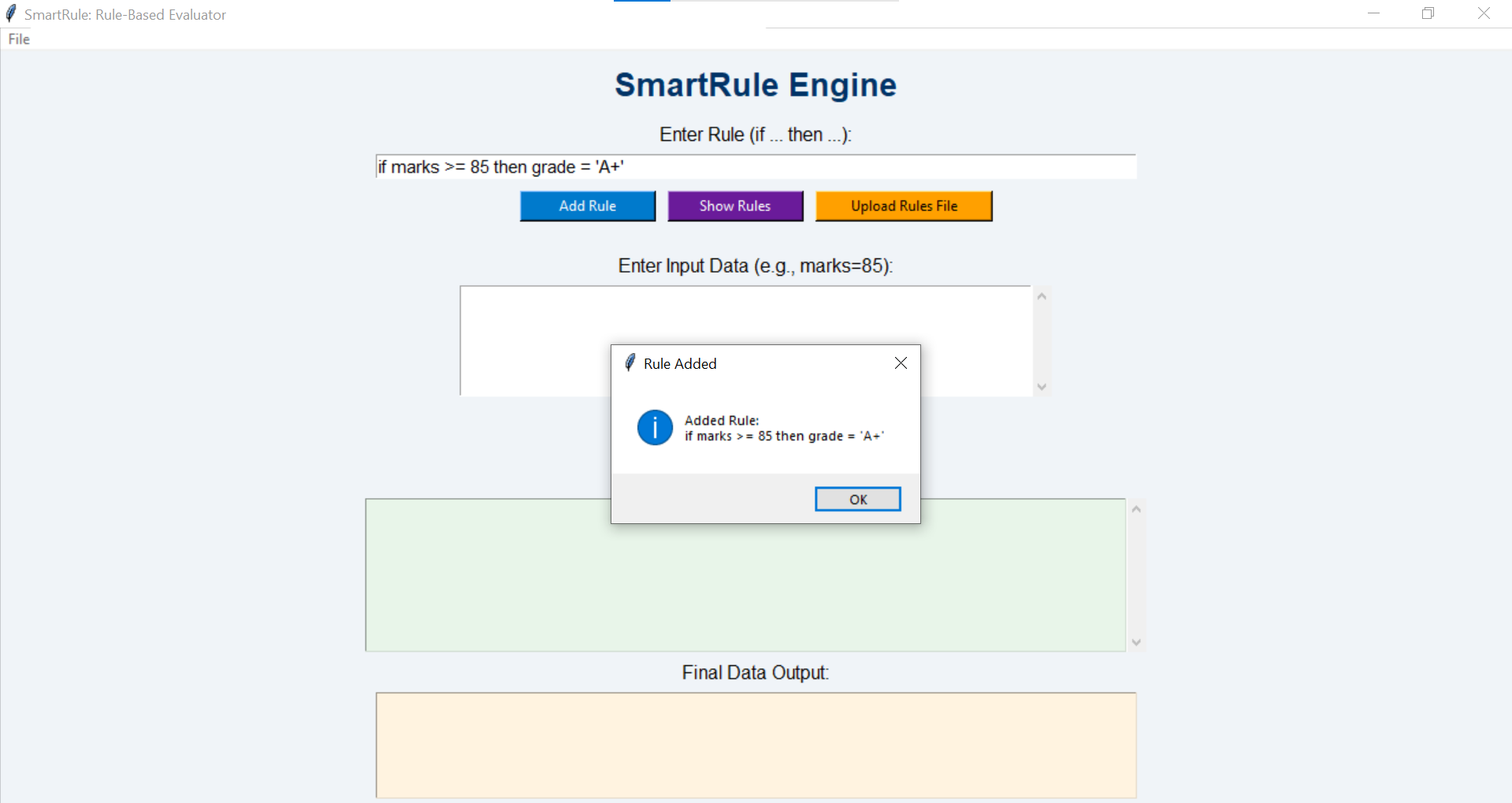
    root.mainloop()

## **Output**

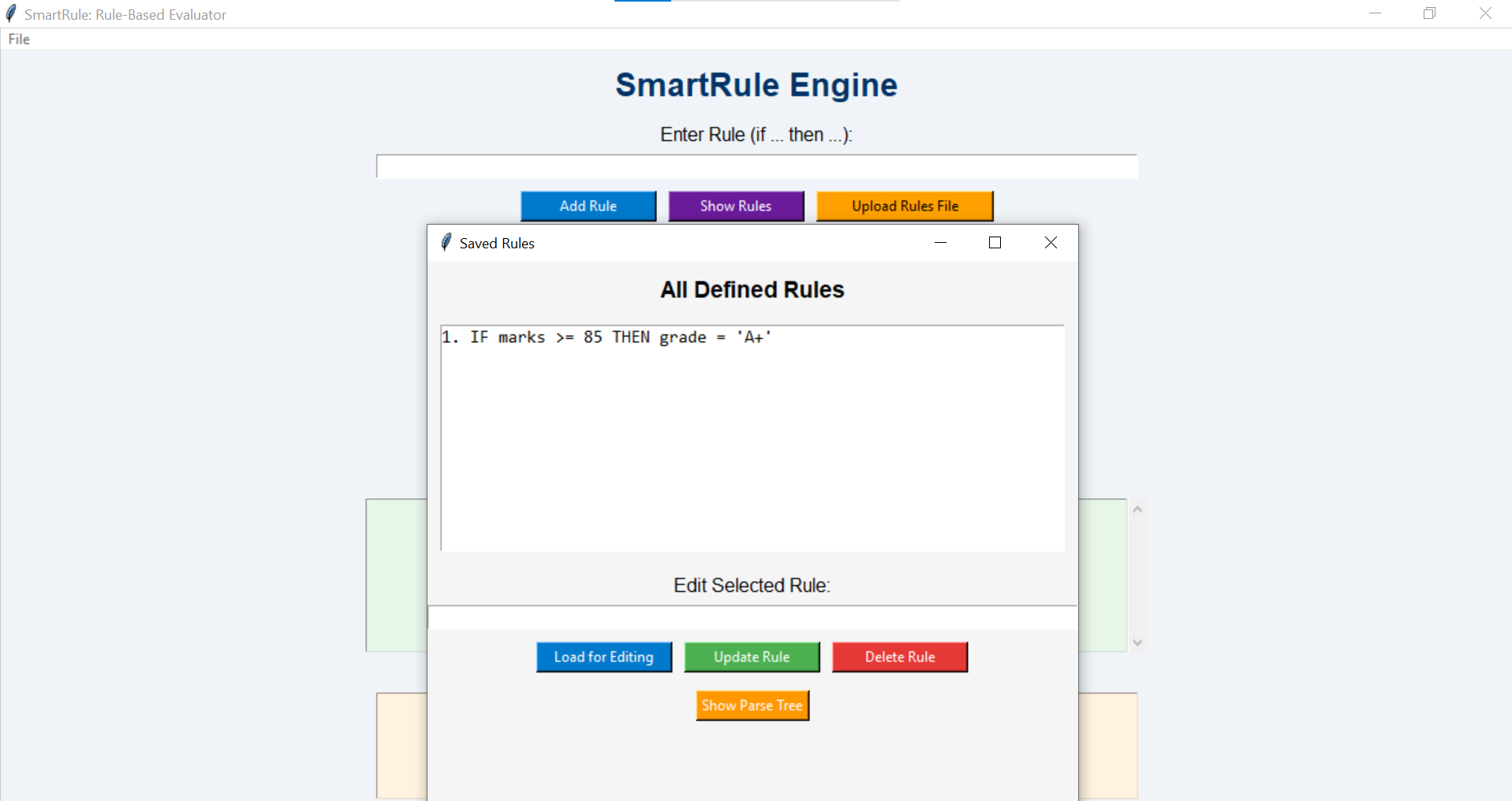
## **Main Page**



## **Add Rule**

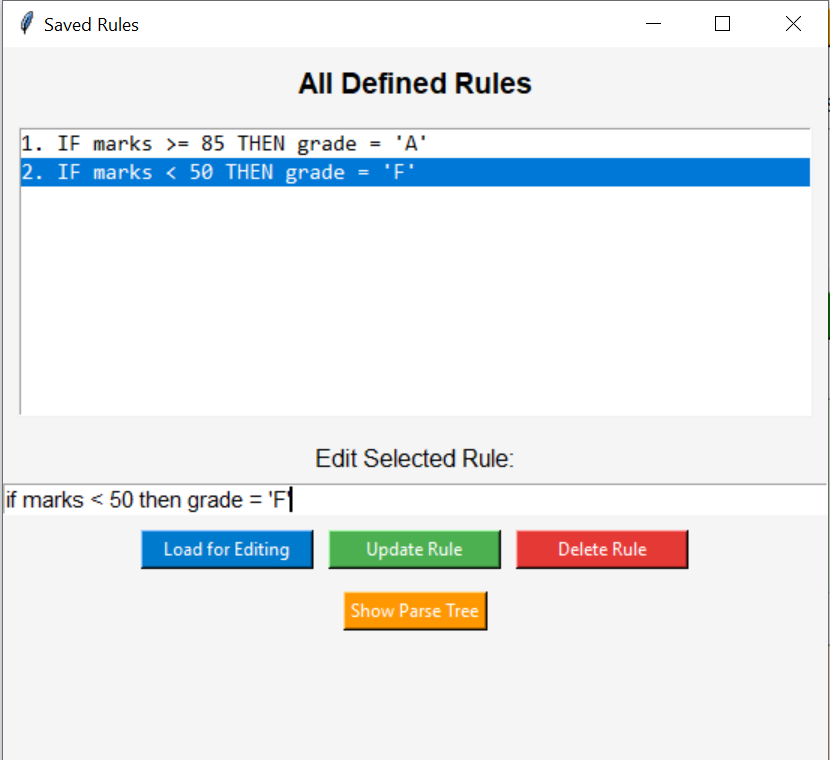
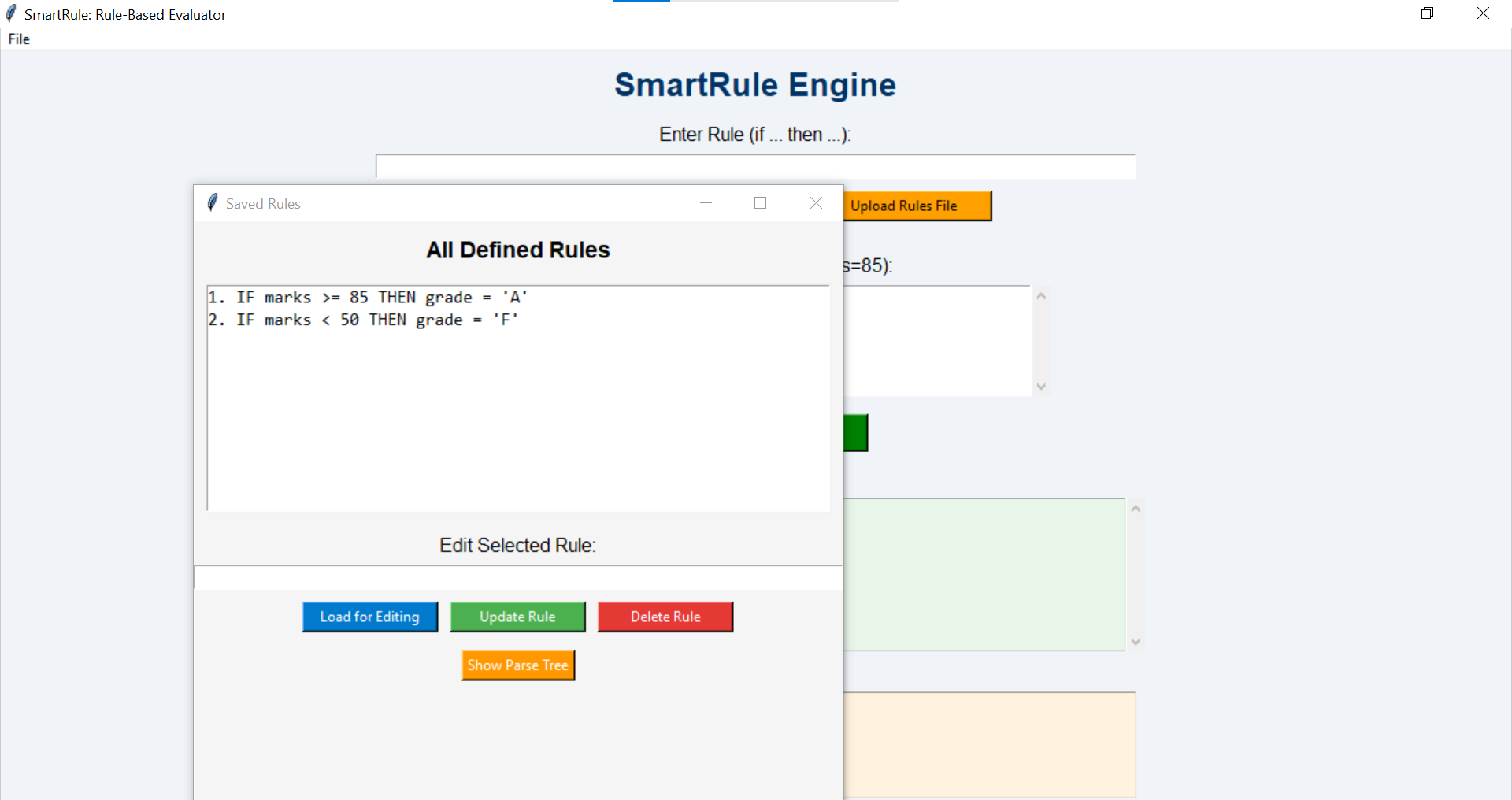


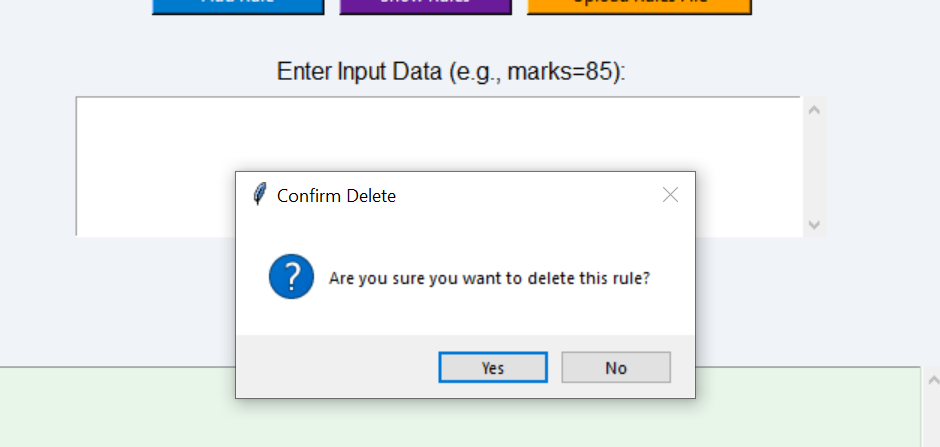
## **Show Rule**

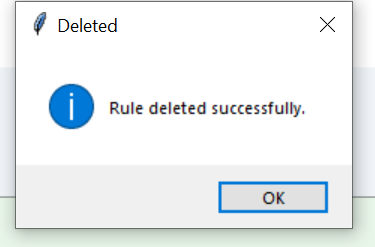


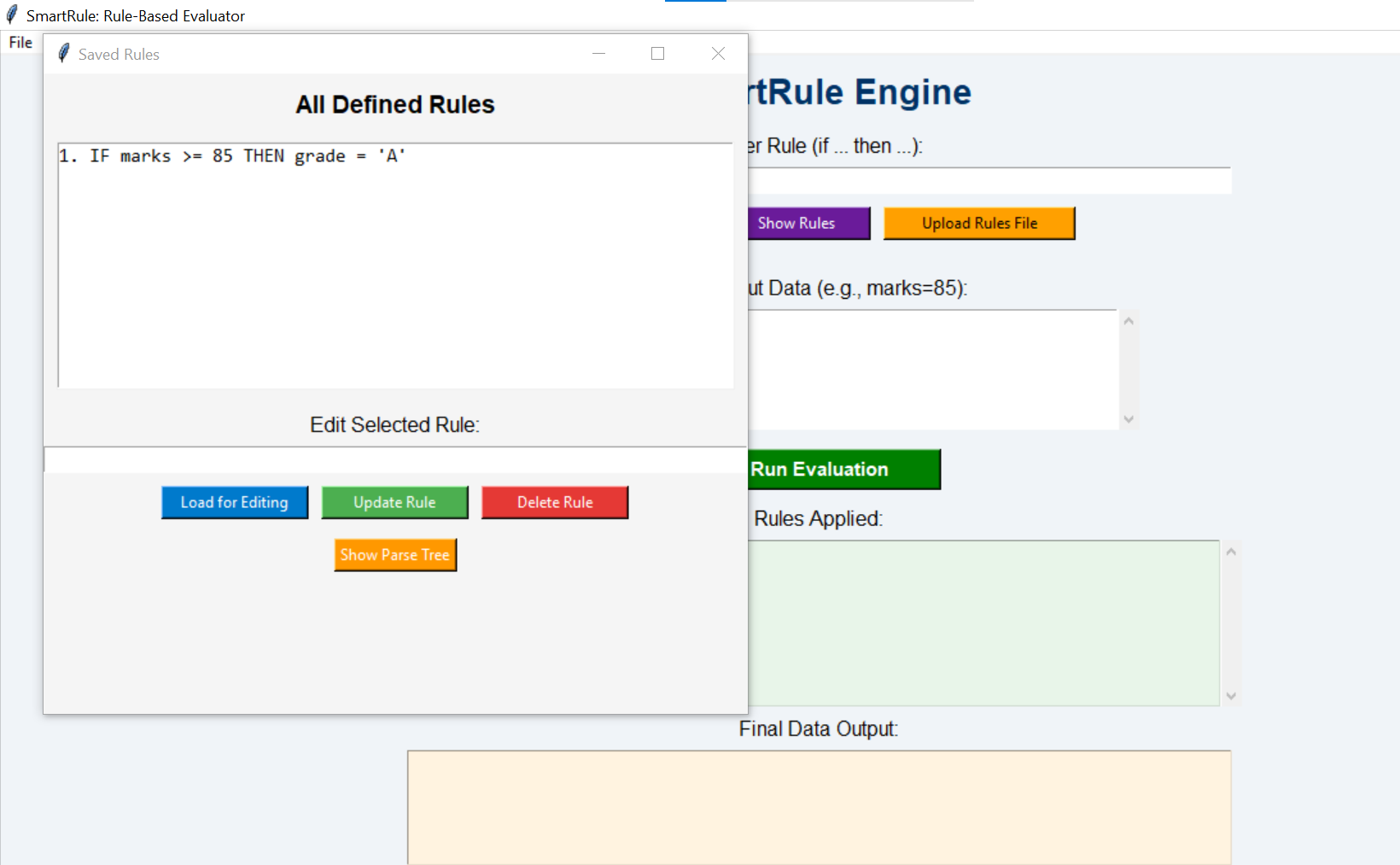
## **Edit Rule**

## **Delete Rule**

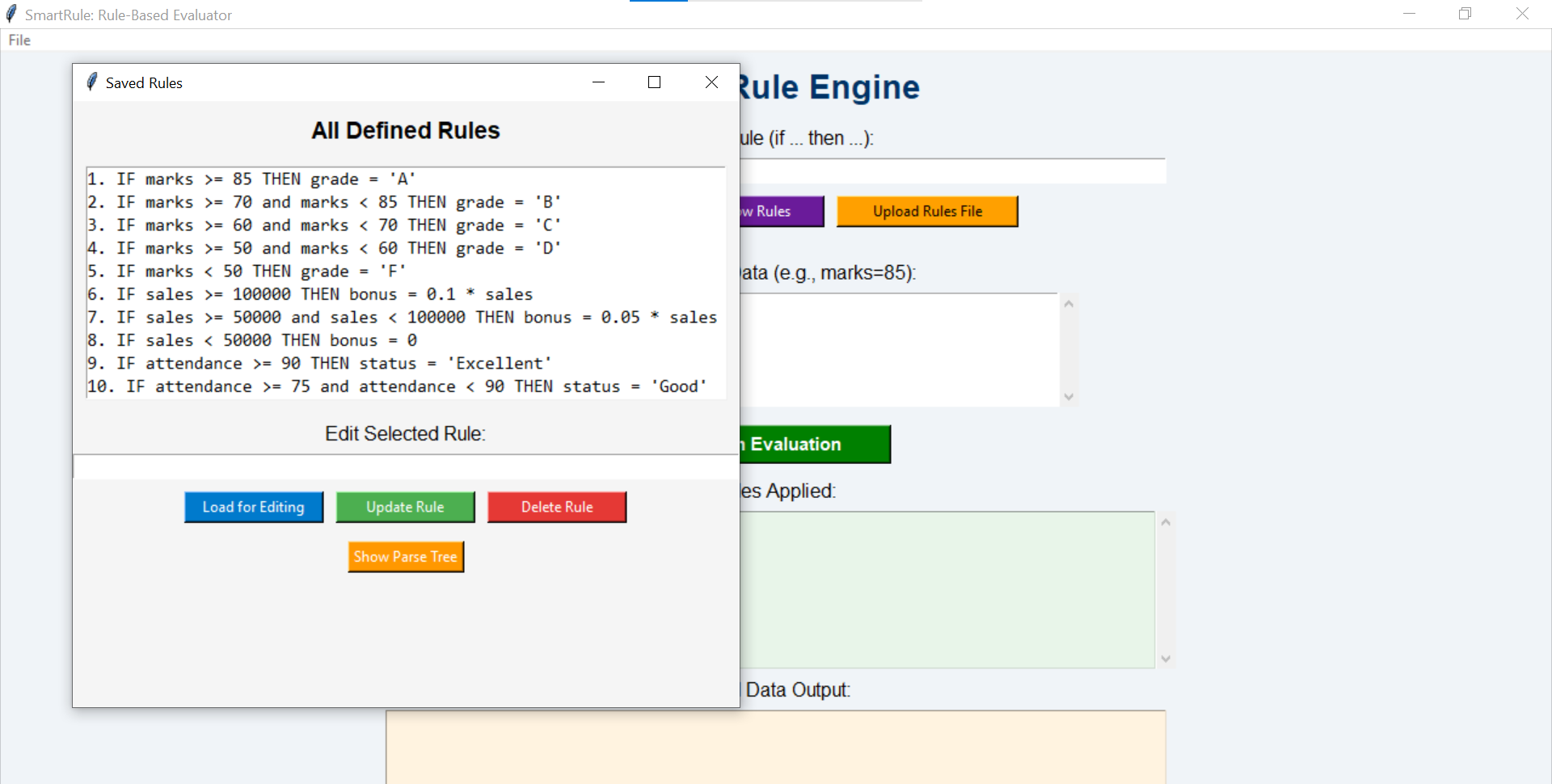
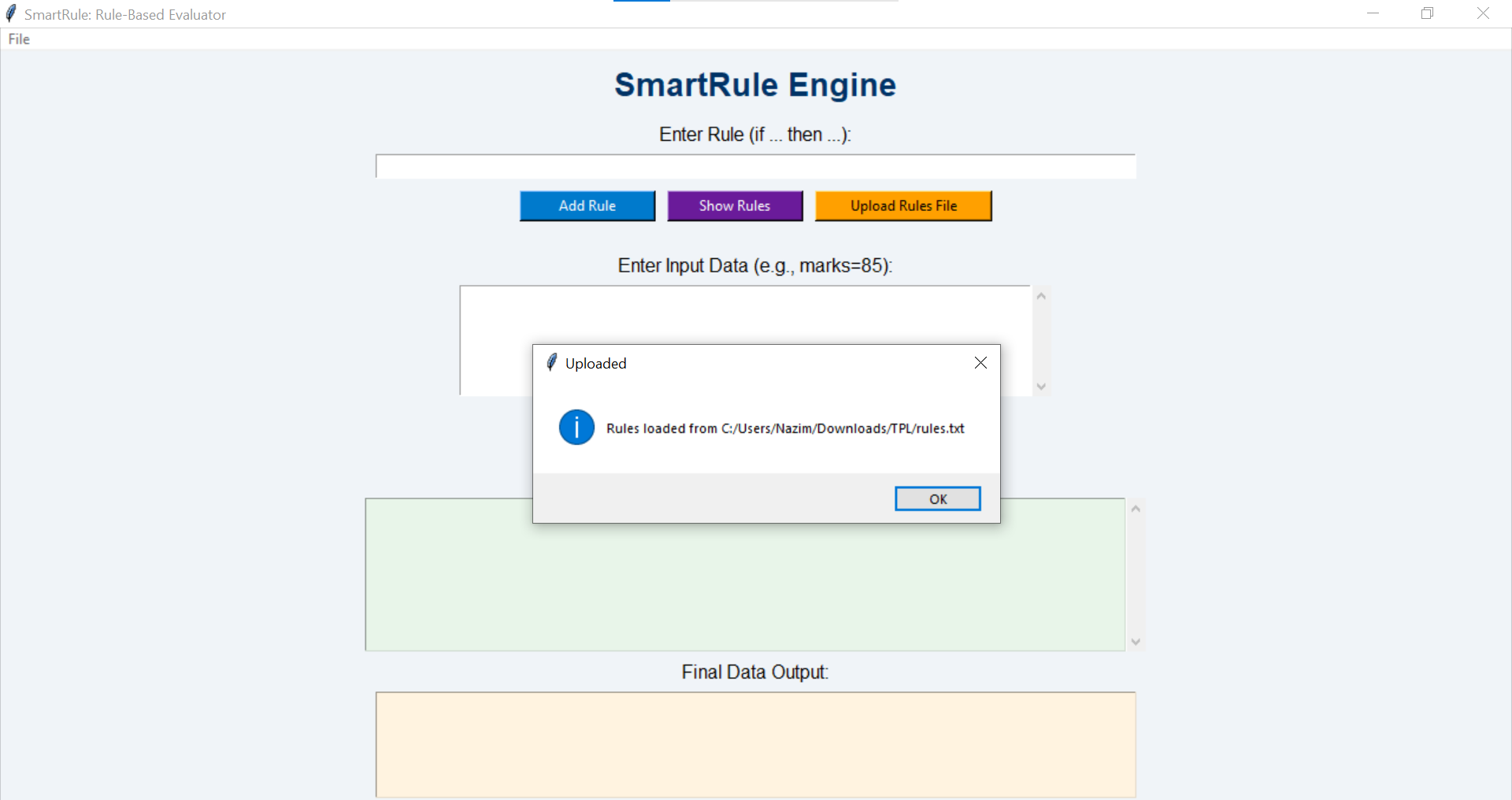
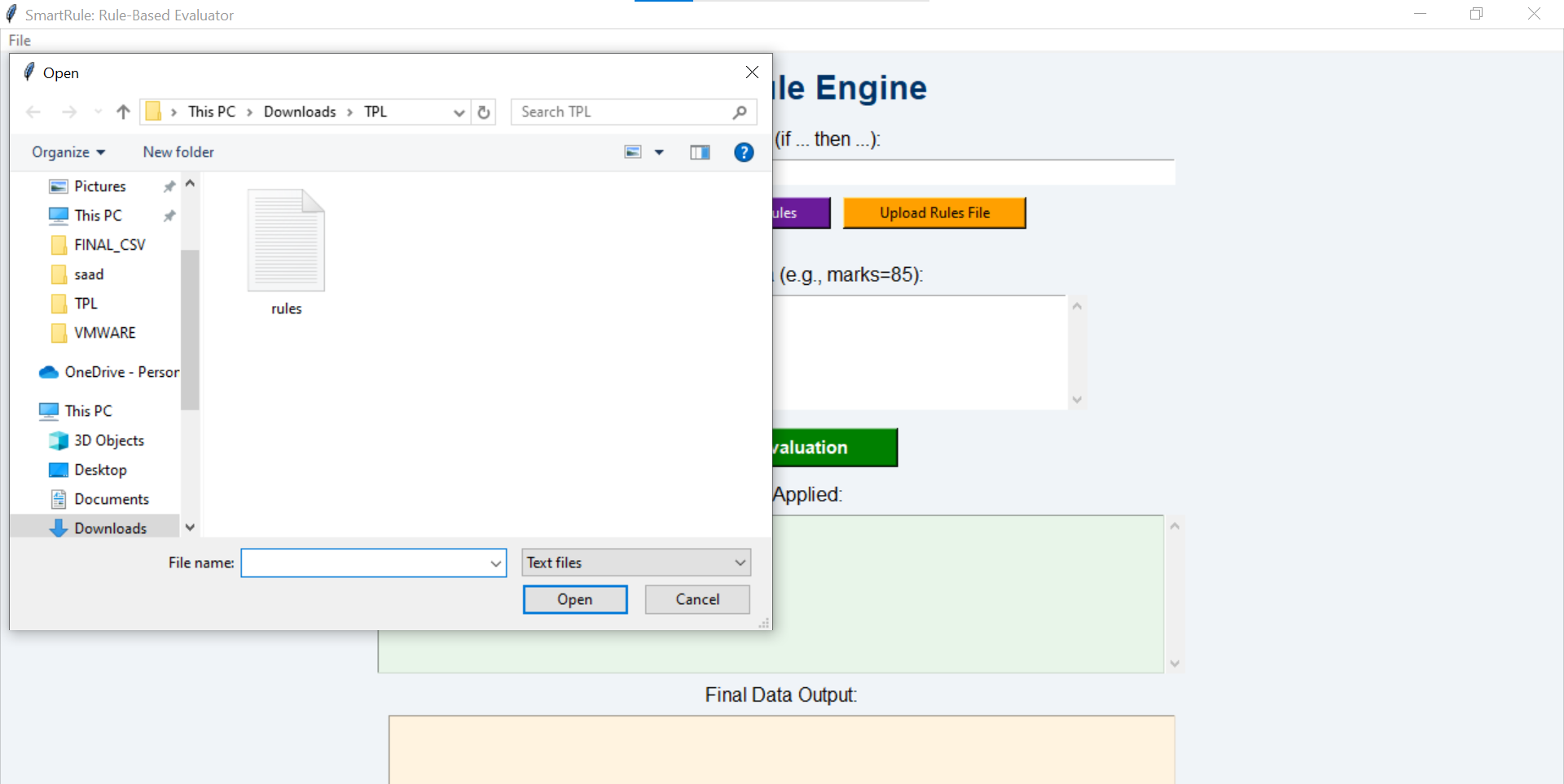




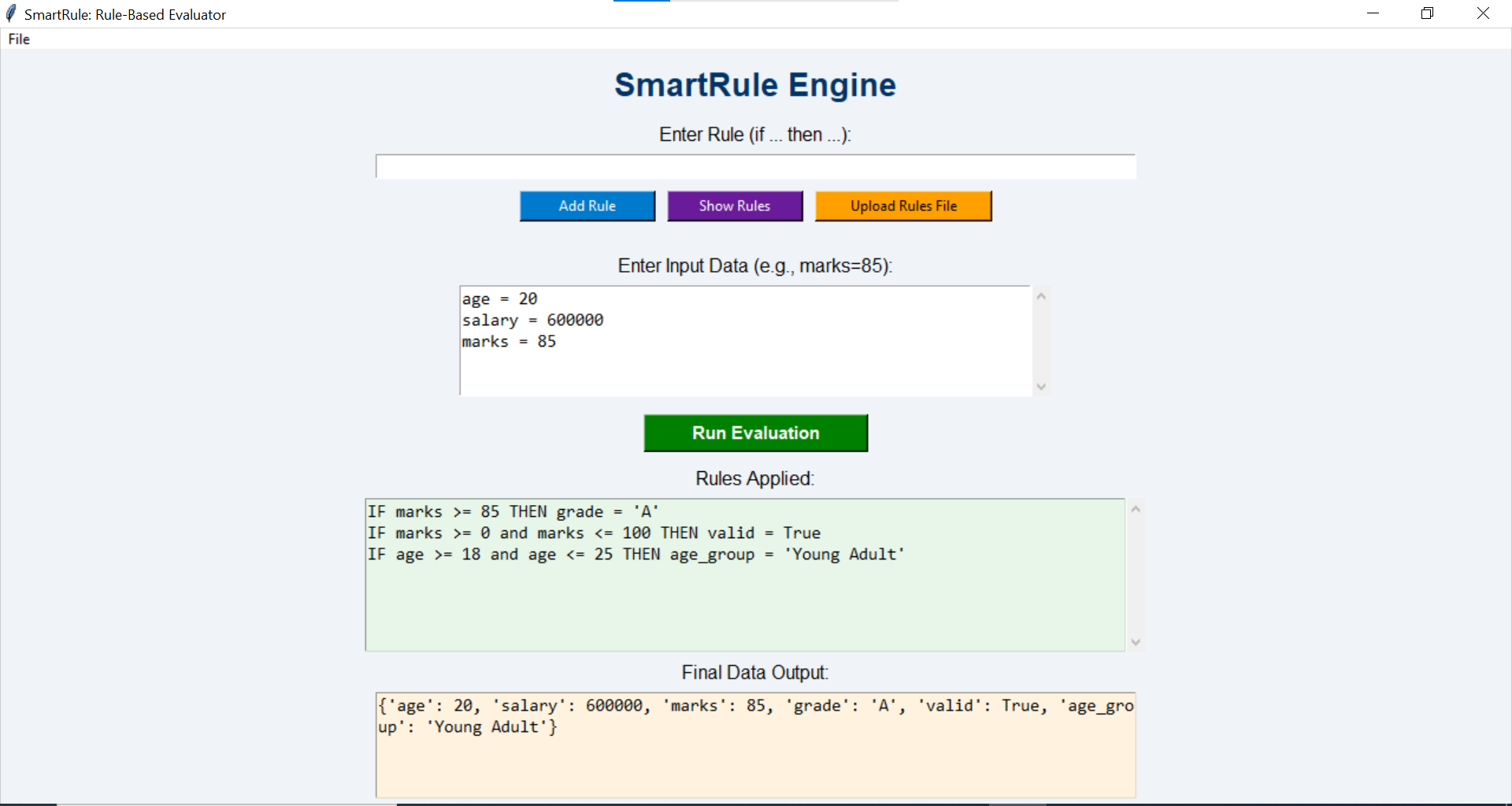




## **Upload Rules**



## **Evaluation**



## **Parse Tree of Rule**

