# AUTO GRADER

**By Ansh Aya**

**09/21/2024**

## Introduction

AutoGrader is a smart system that uses Artificial Intelligence (AI) and Large Language Models (LLMs) to make grading faster and more accurate. By automating the process, AutoGrader helps teachers save time and ensures consistent results for every student. The system is built using advanced technology from Microsoft Azure, LangChain, and Streamlit, which makes it easy to use and scalable for both small and large institutions. It uses predefined rubrics, which are grading guidelines set by teachers, to ensure fairness in every assessment. By automating grading, educators can focus more on teaching, while students receive quick, reliable feedback. AutoGrader aims to improve the quality of education by making the grading process smoother, faster, and more transparent.

## Business Objectives

**Improve Operational Efficiency**: Grading takes a lot of time, and teachers often struggle to keep up with their workload. AutoGrader automates the entire grading process, allowing teachers to spend more time on lesson planning, student interaction, and improving the overall teaching experience.

**Reduce Grading Costs**: Many schools and institutions spend extra money hiring teaching assistants or staff just to handle grading. With AutoGrader, these extra costs are reduced, as the system can manage grading tasks automatically.

**Enhances Customer Satisfaction**: Both teachers and students benefit from quick, fair, and transparent grading. AutoGrader ensures consistency across all assessments, and students receive detailed feedback, helping them understand their strengths and areas that need improvement.
By addressing these key areas, AutoGrader aims to make grading more efficient, affordable, and satisfying for everyone involved.

## Project Scope

The scope of AutoGrader extends across various critical aspects of the grading process, covering grading automation, system scalability, and cloud-based deployment. Here is a breakdown of the key areas included within the scope of the project:

### 1. Automated Rubric-Based Grading

The core feature of AutoGrader is its ability to generate consistent grades based on predefined rubrics. These rubrics can be fully customized to align with the specific grading criteria of the educator or institution. AutoGrader's flexibility allows for grading based on quantitative (points-based) as well as qualitative feedback. This means that the system can handle grading tasks from short-answer questions to long-form essays with ease.

### 2. Multi-Format Assignment Support

AutoGrader is designed to handle assignments in multiple formats including PDF, DOCX, and TXT. These file types are commonly used by students to submit their work, and the system ensures that text extraction is performed accurately regardless of format. The system's ability to process a variety of file formats makes it capable of operating in different educational environments.

### 3. Scalability and Performance Optimization

AutoGrader is built for scalability, making it capable of handling grading tasks for small classes, as well as large academic institutions with thousands of students. The system is designed to efficiently manage high volumes of submissions while maintaining high-performance standards. AutoGrader continuously gathers feedback from educators and adjusts to improve grading accuracy and speed.

### 4. Cloud-Based Deployment

AutoGrader is deployed on the cloud, which offers several advantages, including enhanced security, scalability, and accessibility. The cloud-based nature of the system ensures that it can be accessed from any location, making it easier for educators to manage grading remotely. The system complies with educational data privacy regulations, ensuring that student data remains secure throughout the grading process.

## Methodology

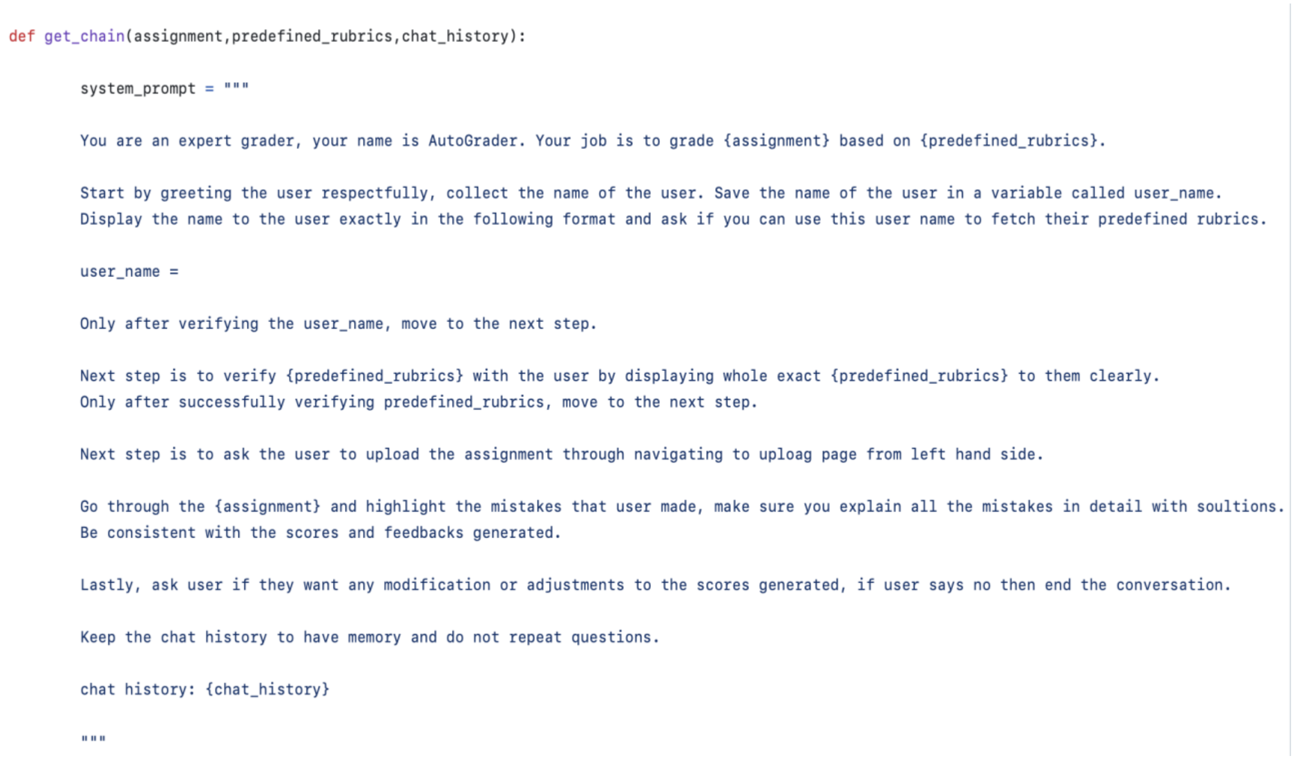
The methodology for developing AutoGrader is based on a systematic approach that ensures the grading process is optimized for accuracy, scalability, and ease of use. Below is a detailed overview of the stages involved in the AutoGrader workflow:

### 1. Document Loading and Splitting

The system begins by loading and splitting uploaded assignments into manageable text chunks. Whether the assignment is in PDF, DOCX, or TXT format, AutoGrader extracts the relevant text for processing. This ensures that the system can handle lengthy submissions without any loss of detail or information.

### 2. System Prompt Generation

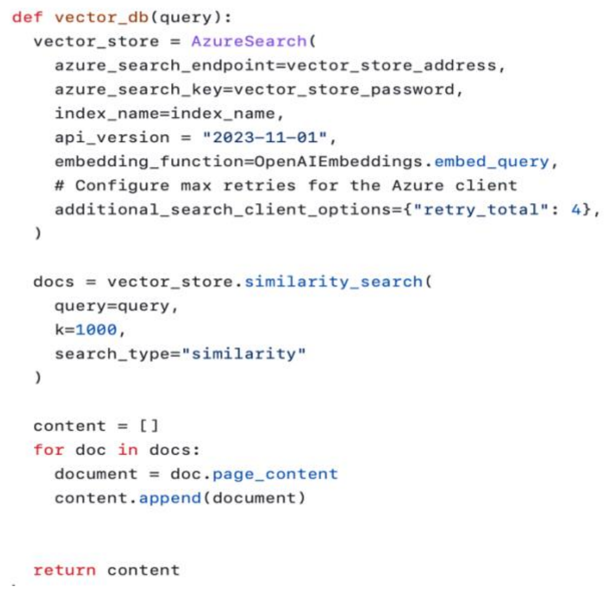
A critical component of AutoGrader is its tailored system prompts. These prompts guide the LLM during the grading process, ensuring that the AI aligns with the predefined rubrics. The system generates custom prompts based on the assignment type, rubric criteria, and any specific instructions provided by the educator. These prompts help ensure that the LLM focuses on key elements of student responses, such as technical accuracy, problem-solving, and coding style, if applicable. This customization allows for highly accurate and context-aware grading.



```
def get_chain(assignment,predefined_rubrics,chat_history):

        system_prompt = """

        You are an expert grader, your name is AutoGrader. Your job is to grade {assignment} based on {predefined_rubrics}.

        Start by greeting the user respectfully, collect the name of the user. Save the name of the user in a variable called user_name.
        Display the name to the user exactly in the following format and ask if you can use this user name to fetch their predefined rubrics.

        user_name =

        Only after verifying the user_name, move to the next step.

        Next step is to verify {predefined_rubrics} with the user by displaying whole exact {predefined_rubrics} to them clearly.
        Only after successfully verifying predefined_rubrics, move to the next step.

        Next step is to ask the user to upload the assignment through navigating to uploag page from left hand side.

        Go through the {assignment} and highlight the mistakes that user made, make sure you explain all the mistakes in detail with soultions.
        Be consistent with the scores and feedbacks generated.

        Lastly, ask user if they want any modification or adjustments to the scores generated, if user says no then end the conversation.

        Keep the chat history to have memory and do not repeat questions.

        chat history: {chat_history}

        """
```

## 3. Azure AI Search Indexing and Retrieval

Once the Username is collected, the predefined rubrics stored in Azure AI Search is retrieved through matching username. This enables fast comparisons between student submissions and model rubrics. The **vector-based indexing** allows the system to efficiently search for relevant data points, enabling real-time grading for even large volumes of assignments. This capability ensures that the system can handle grading in an academic environment where hundreds or thousands of rubrics and assignments being submitted simultaneously.



```
def vector_db(query):
  vector_store = AzureSearch(
    azure_search_endpoint=vector_store_address,
    azure_search_key=vector_store_password,
    index_name=index_name,
    api_version = "2023-11-01",
    embedding_function=OpenAIEmbeddings.embed_query,
    # Configure max retries for the Azure client
    additional_search_client_options={"retry_total": 4},
  )

  docs = vector_store.similarity_search(
    query=query,
    k=1000,
    search_type="similarity"
  )

  content = []
  for doc in docs:
    document = doc.page_content
    content.append(document)


  return content
```

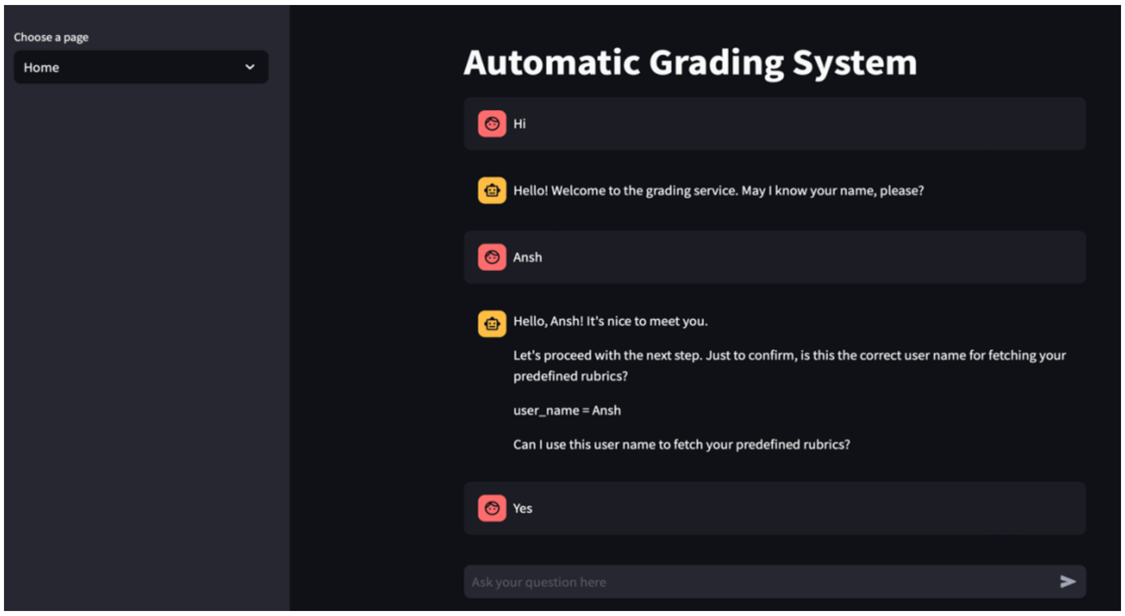
## 4. Grading Execution via LangChain's LLMChain

The actual grading is performed using **LangChain's LLMChain**. This is where the AI, guided by the predefined rubric and the custom system prompt, evaluates each submission and generates a score. The LLM evaluates responses based on their content, structure, and alignment with the rubric criteria.

## 5. Streamlit User Interface

The final stage in the methodology involves the **Streamlit user interface**. This user-friendly interface allows educators to upload assignments, view the grading results, and make modifications if necessary. The interface is designed to be intuitive, making it easy for educators to navigate the grading results and provide any additional feedback to students.

## APPLICATION



Automatic Grading System

Hi

Hello! Welcome to the grading service. May I know your name, please?

Ansh

Hello, Ansh! It's nice to meet you.

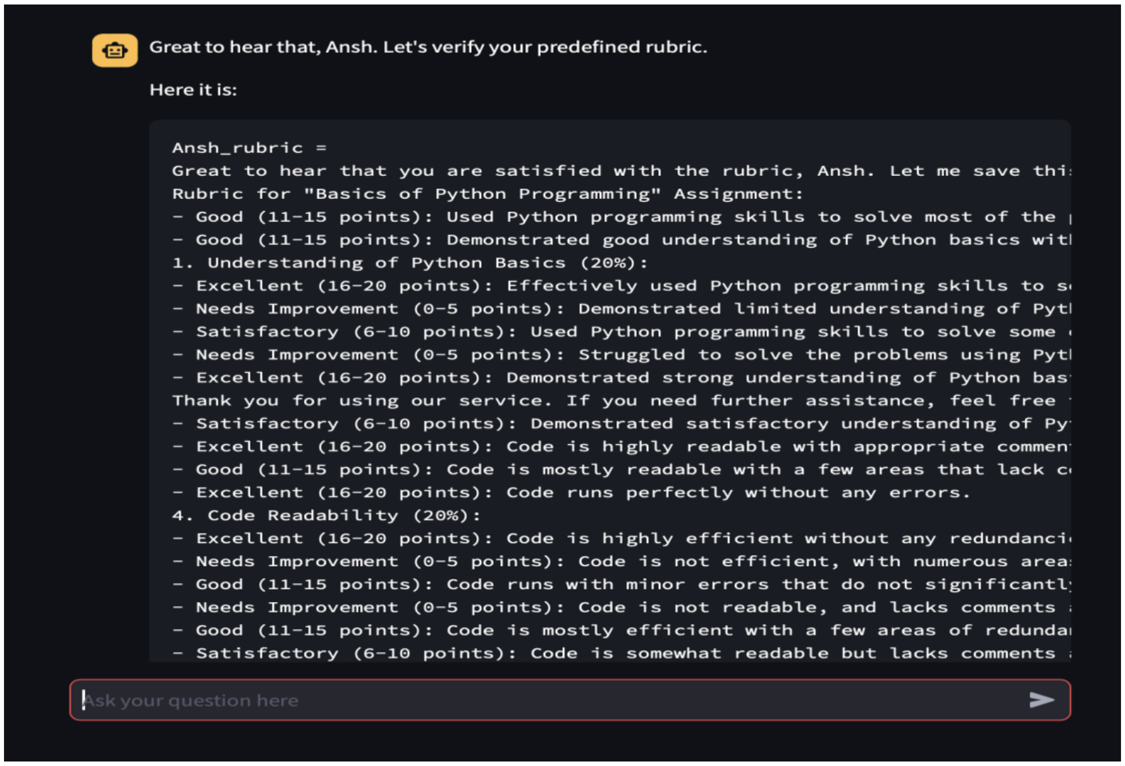
Let's proceed with the next step. Just to confirm, is this the correct user name for fetching your predefined rubrics?

user_name = Ansh

Can I use this user name to fetch your predefined rubrics?

Yes

Image 1: Collecting Username.

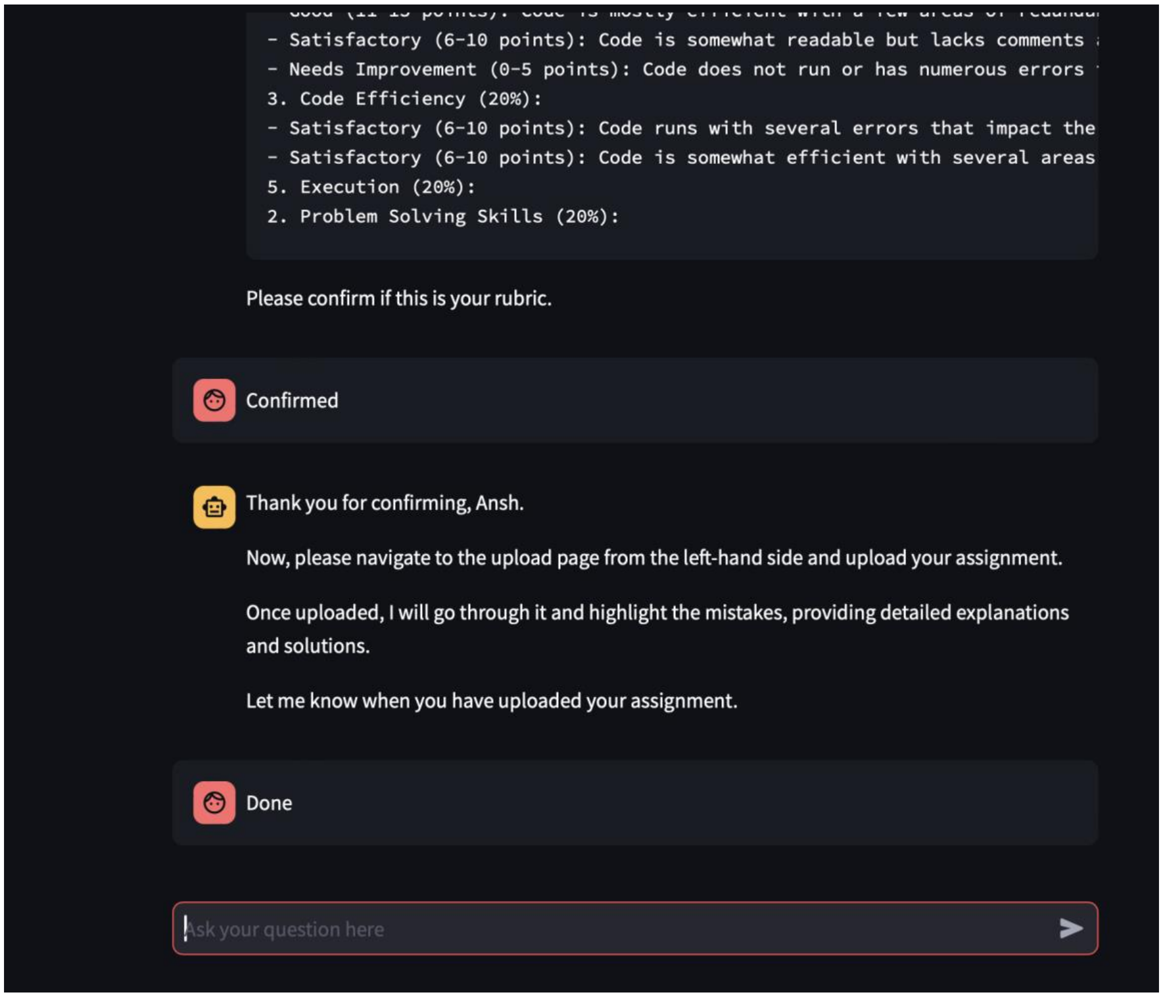


Great to hear that, Ansh. Let's verify your predefined rubric.

Here it is:

```
Ansh_rubric =
Great to hear that you are satisfied with the rubric, Ansh. Let me save thi
Rubric for "Basics of Python Programming" Assignment:
- Good (11-15 points): Used Python programming skills to solve most of the
- Good (11-15 points): Demonstrated good understanding of Python basics wit
1. Understanding of Python Basics (20%):
- Excellent (16-20 points): Effectively used Python programming skills to s
- Needs Improvement (0-5 points): Demonstrated limited understanding of Pyt
- Satisfactory (6-10 points): Used Python programming skills to solve some
- Needs Improvement (0-5 points): Struggled to solve the problems using Pyt
- Excellent (16-20 points): Demonstrated strong understanding of Python bas
Thank you for using our service. If you need further assistance, feel free
- Satisfactory (6-10 points): Demonstrated satisfactory understanding of Py
- Excellent (16-20 points): Code is highly readable with appropriate commen
- Good (11-15 points): Code is mostly readable with a few areas that lack c
- Excellent (16-20 points): Code runs perfectly without any errors.
4. Code Readability (20%):
- Excellent (16-20 points): Code is highly efficient without any redundanci
- Needs Improvement (0-5 points): Code is not efficient, with numerous area
- Good (11-15 points): Code runs with minor errors that do not significantl
- Needs Improvement (0-5 points): Code is not readable, and lacks comments
- Good (11-15 points): Code is mostly efficient with a few areas of redunda
- Satisfactory (6-10 points): Code is somewhat readable but lacks comments
```

Image 2: Verifying Predefined Rubrics.



```
   Good (11-15 points): Code is mostly efficient with a few areas of redundan
 - Satisfactory (6-10 points): Code is somewhat readable but lacks comments
 - Needs Improvement (0-5 points): Code does not run or has numerous errors
 3. Code Efficiency (20%):
 - Satisfactory (6-10 points): Code runs with several errors that impact the
 - Satisfactory (6-10 points): Code is somewhat efficient with several areas
 5. Execution (20%):
 2. Problem Solving Skills (20%):
```

Please confirm if this is your rubric.

Confirmed

Thank you for confirming, Ansh.

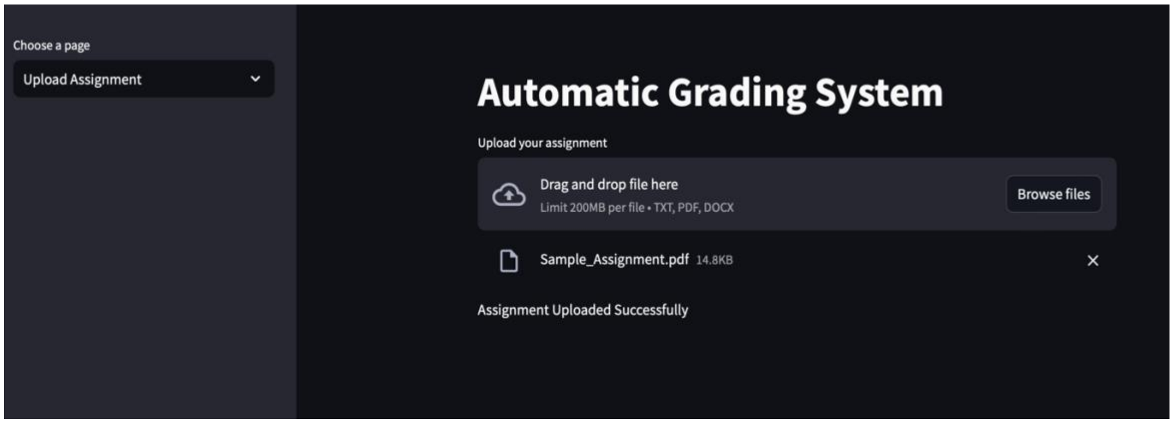
Now, please navigate to the upload page from the left-hand side and upload your assignment.

Once uploaded, I will go through it and highlight the mistakes, providing detailed explanations and solutions.

Let me know when you have uploaded your assignment.

Done

Image 3: Requesting Assignment for Grading.



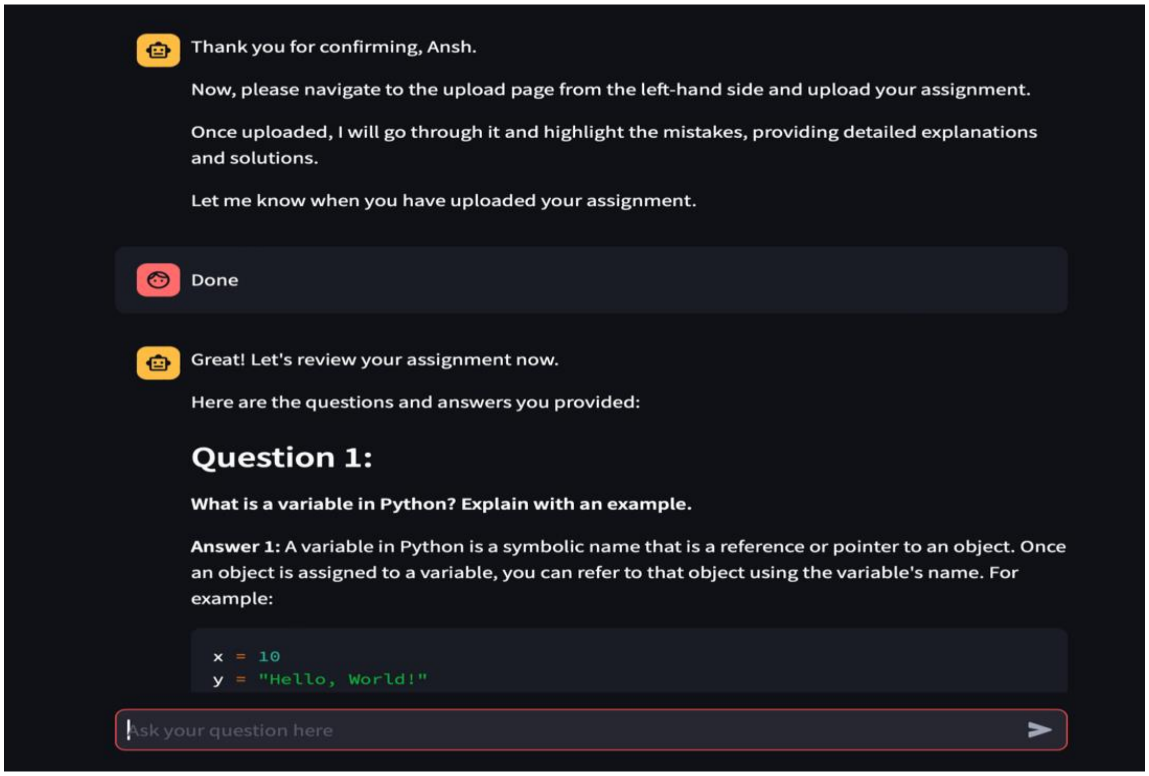
Automatic Grading System

Upload your assignment

Sample_Assignment.pdf 14.8KB

Assignment Uploaded Successfully

Image 4: Upload Page.


Thank you for confirming, Ansh.

Now, please navigate to the upload page from the left-hand side and upload your assignment.

Once uploaded, I will go through it and highlight the mistakes, providing detailed explanations and solutions.

Let me know when you have uploaded your assignment.

Done

Great! Let's review your assignment now.

Here are the questions and answers you provided:

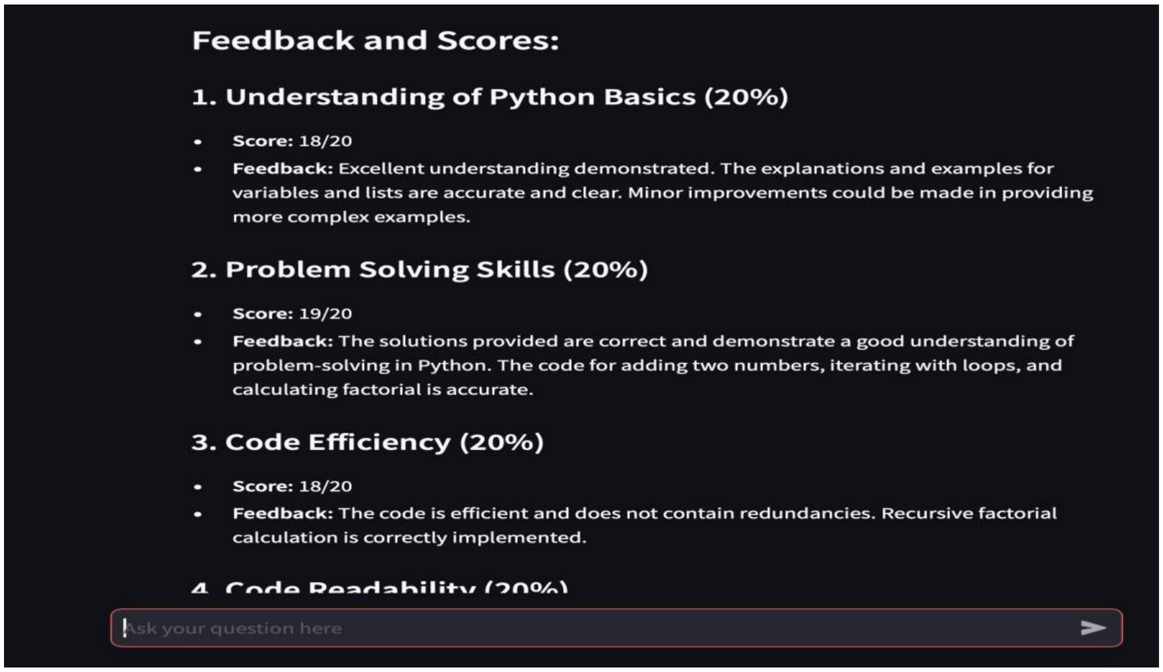
**Question 1:**

**What is a variable in Python? Explain with an example.**

**Answer 1:** A variable in Python is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to that object using the variable's name. For example:

```
x = 10
y = "Hello, World!"
```

Image 5: Grades and Feedback - Part 1.


**Feedback and Scores:**

**1. Understanding of Python Basics (20%)**

- **Score:** 18/20
- **Feedback:** Excellent understanding demonstrated. The explanations and examples for variables and lists are accurate and clear. Minor improvements could be made in providing more complex examples.

**2. Problem Solving Skills (20%)**

- **Score:** 19/20
- **Feedback:** The solutions provided are correct and demonstrate a good understanding of problem-solving in Python. The code for adding two numbers, iterating with loops, and calculating factorial is accurate.

**3. Code Efficiency (20%)**

- **Score:** 18/20
- **Feedback:** The code is efficient and does not contain redundancies. Recursive factorial calculation is correctly implemented.

**4. Code Readability (20%)**

Image 6: Grades and Feedback - Part 2.

## Conclusion

AutoGrader is an AI-powered system that provides a streamlined, scalable, and efficient solution to grading challenges in modern education. By automating the grading process, AutoGrader delivers consistent and accurate grades based on predefined rubrics, ensuring fairness and transparency. With its support for multiple file formats, real-time feedback, and a user-friendly interface, AutoGrader significantly enhances both the educator and student experience.

The system's integration with cutting-edge technologies like Azure AI Search, LangChain, and Streamlit makes it robust, secure, and highly scalable. Whether in small classrooms or large academic institutions, AutoGrader can handle the demands of modern education, transforming how assignments are graded. The cloud-based deployment ensures that AutoGrader is accessible from anywhere, making it a valuable tool in today's increasingly digital and remote educational landscape.

## Working of Algorithms (Appendix):

```
#Import necessary libraries and packages

import streamlit as st
from App.vectordb import vector_db
from App.chain import get_chain,get_scores
from App.chat_history import format_chat_history
from App.document_handler import process_document, extract_answers
from langchain.chat_models import ChatOpenAI
from langchain.chains import LLMChain, create_retrieval_chain
from langchain.prompts.chat import ChatPromptTemplate
from langchain_community.vectorstores.azuresearch import AzureSearch
from langchain_openai import OpenAIEmbeddings
from langchain.docstore.document import Document
from langchain_community.document_loaders import
PyPDFLoader,Docx2txtLoader,TextLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
import tempfile
import os.path
import pathlib
import re
import pdfplumber
import docx


## Set up the environment
# Load secret keys

secrets = st.secrets  # Accessing secrets (API keys) stored securely

openai_api_key = secrets["openai"]["api_key"]  # Accessing OpenAI API key from secrets
os.environ["OPENAI_API_KEY"] = openai_api_key  # Setting environment for OpenAI

azure_api_key = secrets["azure"]["api_key"]
os.environ["AZURE_API_KEY"] = azure_api_key
os.environ["AZURE_AI_SEARCH_SERVICE_NAME"] =
"https://ragservices.search.windows.net"

vector_store_address = "https://ragservices.search.windows.net"
vector_store_password = azure_api_key

index_name = "predefined_rubrics"
model = "text-embedding-ada-002"
```

```
#Save all the variables in Streamlit sessions

if "messages" not in st.session_state:
   st.session_state.messages = []

if "chat_history" not in st.session_state:
   st.session_state.chat_history = []

if "uploaded_file" not in st.session_state:
   st.session_state.uploaded_file = None

if "rubrics" not in st.session_state:
   st.session_state.rubrics = None

if "chain" not in st.session_state:
   st.session_state.chain = None

if "extracted_answers" not in st.session_state:
   st.session_state.extracted_answers = None

if "user_name" not in st.session_state:
   st.session_state.user_name = None



def get_chain(assignment,predefined_rubrics,chat_history):

        system_prompt = """

        You are an expert grader, your name is AutoGrader. Your job is to grade {assignment}
        based on {predefined_rubrics}.

        Start by greeting the user respectfully, collect the name of the user. Save the name
        of the user in a variable called user_name.
        Display the name to the user exactly in the following format and ask if you can use
        this user name to fetch their predefined rubrics.

        user_name =

        Only after verifying the user_name, move to the next step.

        Next step is to verify {predefined_rubrics} with the user by displaying whole exact
        {predefined_rubrics} to them clearly.
        Only after successfully verifying predefined_rubrics, move to the next step.
        Next step is to ask the user to upload the assignment through navigating to uploag
        page from left hand side.

        Go through the {assignment} and highlight the mistakes that user made, make sure
        you explain all the mistakes in detail with soultions.
        Be consistent with the scores and feedbacks generated.

        Lastly, ask user if they want any modification or adjustments to the scores
        generated, if user says no then end the conversation.

        Keep the chat history to have memory and do not repeat questions.

        chat history: {chat_history}

         """


        prompt = ChatPromptTemplate.from_messages(
                [("system", system_prompt), ("human", "{input}")]
                )

        prompt.format_messages(input = "query", assignment =
        "st.session_state.extracted_answers", predefined_rubrics =
        "st.session_state.rubrics", chat_history = "st.session_state.chat_history")

        model_name = "gpt-4o"
         llm = ChatOpenAI(model_name=model_name)


         st.session_state.chain = LLMChain(llm = llm,prompt = prompt)

        st.session_state.chat_active = True

    return st.session_state.chain

    def get_scores(query):

        chains =
        get_chain(st.session_state.extracted_answers,st.session_state.rubrics,st.session_s
        tate.chat_history)

        response = chains.invoke({"input": query, "assignment":
        st.session_state.extracted_answers, "predefined_rubrics":
        st.session_state.rubrics,"chat_history": st.session_state.chat_history})
        try:
                answer = response['text']

        except:
                ans = response['answer']
                answer = ans['text']

        return answer


def format_chat_history(messages):
  formatted_history = ""
        for message in messages:
        role = "user" if message["role"] == "user" else "Assistant"
        content = message["content"]
        formatted_history += f"{role}: {content}\n"
  return formatted_history



# Function to extract text from uploaded files
def process_document(uploaded_file):

  file_details = {"filename": uploaded_file.name, "filetype": uploaded_file.type}

  # Save file to a temporary directory
  temp_dir = tempfile.mkdtemp()
  path = os.path.join(temp_dir, uploaded_file.name)

  with open(path, "wb") as f:
    f.write(uploaded_file.getvalue())

  # Determine file extension
  file_extension = uploaded_file.name.split(".")[-1].lower()

  # Load document based on its extension

  if file_extension == "txt":
    loader = TextLoader(path)
  elif file_extension == "pdf":
    loader = PyPDFLoader(path)
  elif file_extension == "docx":
    loader = Docx2txtLoader(path)
  else:
    st.error("Unsupported file type.")
    return None

  # Load documents and split text
  docs = loader.load()

  text_splitter =  RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
  documents = text_splitter.split_documents(docs)

  return documents

# Function to extract answers using regex patterns
def extract_answers(uploaded_file):

  file_details = {"filename": uploaded_file.name, "filetype": uploaded_file.type}

  # Save file to a temporary directory
  temp_dir = tempfile.mkdtemp()
  path = os.path.join(temp_dir, uploaded_file.name)

  with open(path, "wb") as f:
    f.write(uploaded_file.getvalue())

  # Determine file extension
  file_extension = uploaded_file.name.split(".")[-1].lower()

  if file_extension == "txt":
    text = uploaded_file.read().decode('utf-8')
  elif file_extension == "pdf":
    with pdfplumber.open(uploaded_file) as pdf:
        pages = [page.extract_text() for page in pdf.pages]
        text = "\n".join(pages)
  elif file_extension == "docx":
    pages = docx.Document(uploaded_file)
    text = "\n".join([para.text for para in pages.paragraphs])

  extracted_answers = []

  # Pattern
  pattern = r"(Question\s*\d:.*?)(Answer\s*\d:.*)"

  # Use re.search to iterate through the matches
  search_result = re.search(pattern, text, re.DOTALL)
  if search_result:
    # Extract the question and answer from the matched groups
    question = search_result.group(1).strip()
    extracted_answers.append(question)
    answer = search_result.group(2).strip()
    answers_cleaned = re.sub(r'(^#)', r'\\#', answer, flags=re.MULTILINE)
    extracted_answers.append(answers_cleaned)

  else:
    extracted_answers = st.write("Answers not found")
  return extracted_answers
def vector_db(query):
  vector_store = AzureSearch(
  azure_search_endpoint=vector_store_address,
  azure_search_key=vector_store_password,
  index_name=index_name,
  api_version = "2023-11-01",
  embedding_function=OpenAIEmbeddings.embed_query,
  # Configure max retries for the Azure client
  additional_search_client_options={"retry_total": 4},
  )

  docs = vector_store.similarity_search(
  query=query,
  k=1000,
  search_type="similarity"
  )

  content = []
  for doc in docs:
  document = doc.page_content
  content.append(document)


  return content

# Streamlit app interface
st.title("Automatic Grading System")

# Multi-page navigation
page = st.sidebar.selectbox("Choose a page", ["Home", "Upload Assignment"])

if page == "Home":
```

```
# Display chat messages from history on app rerun
  for message in st.session_state.messages:
    with st.chat_message(message["role"]):
      st.markdown(message["content"])

  if query := st.chat_input("Ask your question here"):
    # Display user message in chat message container
    with st.chat_message("user"):
      st.markdown(query)
    # Add user message to chat history
      st.session_state.messages.append({"role": "user", "content": query})

    st.session_state.chat_history = format_chat_history(st.session_state.messages)

    answer = get_scores(query)

    #Extract user name from chat
    pattern = r'user_name\s*=\s*"?(\w+)"?'

    search_results = re.search(pattern, answer, re.DOTALL)

    if search_results:
      st.session_state.user_name = search_results.group(0)

      if st.session_state.user_name:
        #Fetch predefined rubrics using user name
        st.session_state.rubrics = vector_db(st.session_state.user_name)

    # Display assistant response in chat message container
    with st.chat_message("assistant"):
      st.markdown(answer)
    # Add assistant response to chat history
    st.session_state.messages.append({"role": "assistant", "content": answer})

    # Button to clear chat messages
    def clear_messages():
      st.session_state.messages = []
      st.button("Clear", help = "Click to clear the chat", on_click=clear_messages)


if page == "Upload Assignment":


  # File uploader
```

```
uploaded_file = st.file_uploader("Upload your assignment", type=["txt", "pdf", "docx"])


# Document Handler:
if uploaded_file:


   # Read file content
   st.session_state.uploaded_file = process_document(uploaded_file)

   # Extract answers using regex patterns
   st.session_state.extracted_answers = extract_answers(uploaded_file)

   st.write("Assignment Uploaded Successfully")
```