Minor in AI

AI-Assisted Programming

1 Case study: Solve a coding problem with AI

Imagine you're a student tasked with creating a complex calendar program. Traditionally, this would require extensive coding knowledge and time. However, with AI-assisted programming, you can accomplish this task efficiently, even with limited coding experience. This real-world scenario illustrates the power and potential of AI in programming education and practice.

2 The Problem and AI Solution

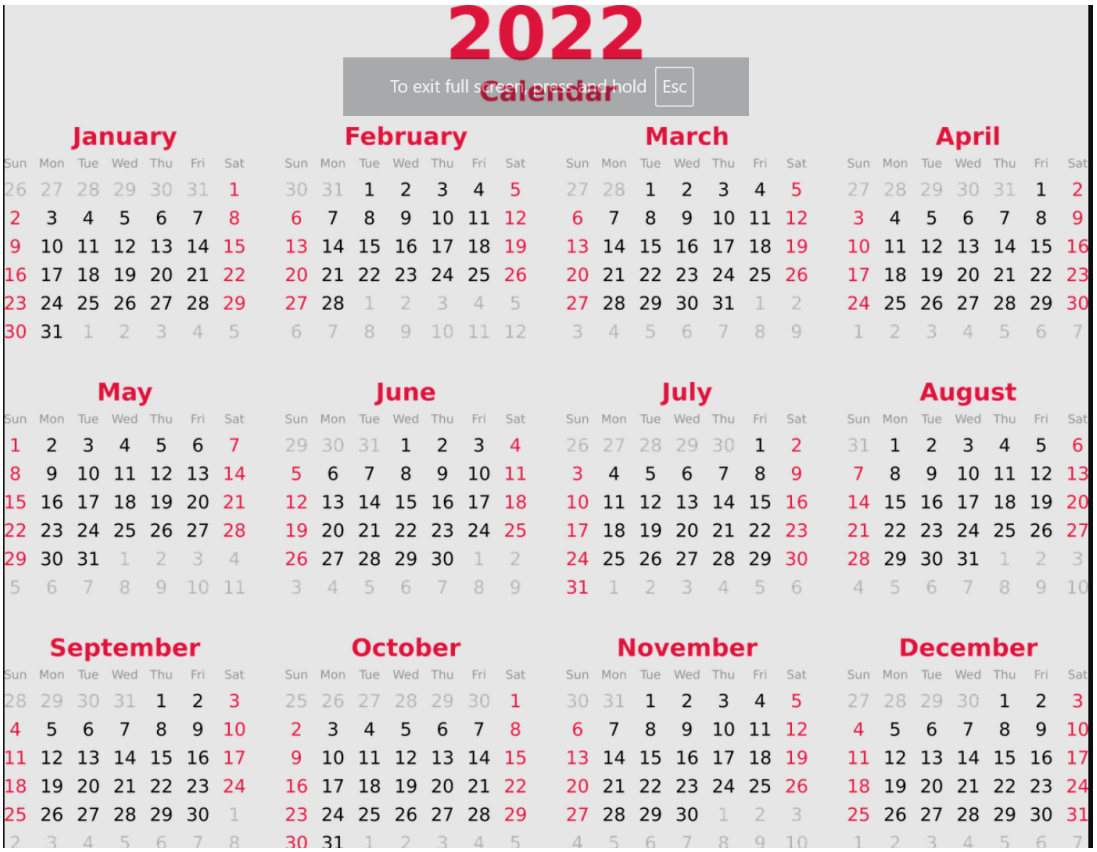
The challenge in programming education is balancing the need for fundamental coding skills with the rapid advancement of AI tools. AI-assisted programming, as demonstrated in this lecture, offers a solution by allowing students to focus on problem-solving and logic while using natural language prompts to generate code.

2.1 Key Concepts

- **AI-Powered Prompts**: Using natural language to generate Python code in Google Colab.
- **Basic Python Programming**: Creating simple programs for tasks like printing numbers and generating calendars.
- **The Birthday Paradox**: Exploring probability through code simulation.

2.2 Demonstration: Calendar Generation

Here's an example of how AI can assist in creating a calendar program:




```python
# AI-generated code based on the prompt:
# "Given a year, display the calendar of that year in a user-friendly way"

import calendar

year = int(input("Enter a year: "))

for month in range(1, 13):
    print(calendar.month(year, month))
```

This code, generated through an AI prompt, demonstrates how complex tasks can be simplified using AI-assisted programming.

3 The Birthday Paradox: A Case Study

The lecture used the Birthday Paradox to illustrate the power of AI-assisted coding in exploring mathematical concepts.

3.1 Problem Statement

In a group of n people, the probability of at least two people sharing the same birthday can be surprisingly high, even for relatively small groups. This phenomenon is known as the **Birthday Paradox**.

3.2 AI-Assisted Solution

We simulated the probability of shared birthdays using Python. Below is the AI-generated code with comments to explain each step:

```
import random

# Function to simulate the birthday paradox
def birthday_paradox(num_people, num_simulations):
    """
    Simulates the birthday paradox and calculates the probability
    of at least two people sharing a birthday.

    Parameters:
    num_people (int): Number of people in the group.
    num_simulations (int): Number of simulations to run.
```

```
12
13      Returns:
14      float: Probability of at least two people sharing a birthday.
15      """
16      matches = 0  # Counter for simulations with shared birthdays
17
18      # Run the simulation num_simulations times
19      for _ in range(num_simulations):
20          # Generate random birthdays for the group
21          birthdays = [random.randint(1, 365) for _ in range(num_people)]
22
23          # Check if there are duplicate birthdays
24          if len(birthdays) != len(set(birthdays)):
25              matches += 1  # Increment matches if duplicates found
26
27      # Return the probability of shared birthdays
28      return matches / num_simulations
29
30  # Example usage of the function
31  result = birthday_paradox(25, 10000)
32  print(f"Probability of a shared birthday in a group of 25: {result:.2f}")
```

3.3 Explanation of the Code

1. **Imports**: The `random` module is used to generate random integers, simulating birthdays.
2. **Function Definition**:
   - `birthday_paradox(num_people, num_simulations)`: This function calculates the probability of shared birthdays.
   - **Parameters**:
     - `num_people`: The size of the group (e.g., 25 people).
     - `num_simulations`: Number of times the simulation is repeated (e.g., 10,000 times for accuracy).
3. **Simulations**:
   - A loop runs the simulation `num_simulations` times.
   - For each simulation, a list of `num_people` random birthdays (integers between 1 and 365) is generated.
   - The function checks for duplicates by comparing the length of the `birthdays` list with the length of the `set(birthdays)` (a set automatically removes duplicates).
4. **Count Matches**:
   - If duplicates are found, the `matches` counter is incremented.

5. **Calculate Probability**:
    - The probability is computed as the ratio of `matches` to `num_simulations`.
6. **Result**:
    - For a group of 25 people, the simulation typically shows that the probability of at least two people sharing a birthday is around 0.57 (57%).

3.4 Key Insight

This code demonstrates the counterintuitive nature of the Birthday Paradox: even with a group as small as 25 people, there is a high likelihood of shared birthdays due to combinatorial probabilities.

4 Conclusion

AI-assisted programming offers a new paradigm in coding education and practice. It allows students to focus on problem-solving and logic while leveraging AI to handle syntax and implementation details. However, it's crucial to balance this with understanding fundamental programming concepts.

Key takeaways:

- AI can significantly speed up the coding process, especially for beginners.
- Understanding the problem and formulating the right prompts is crucial.
- AI-assisted coding can help explore complex concepts like the Birthday Paradox.
- Traditional programming skills remain important for deeper understanding and problem-solving.

5 Additional Resources

For those interested in exploring further:

- Wikipedia: Birthday Paradox
- Library of Babel Website