2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC

Host Sponsor UNC Charlotte

Paper ID #51690

Leveraging Multiple Learning Tools Including a Course-Specific AI Agent, and Weekly Mini-Projects to Foster Critical Thinking and Purposeful Learning in Introductory Programming Courses

Prof. Hasan Baig, University of Connecticut

Dr. Hasan Baig currently serves as an Assistant Professor at the University of Connecticut's Stamford Campus, USA. Prior to joining the Computer Science and Engineering (CSE) department, Dr. Baig was a Postdoctoral Research Fellow in the Mendes Research Group at UConn Health, USA, where he focused on developing cloud-based computational tools for systems biology.

Dr. Baig is award winning educationist who has developed numerous EdTech tools for students and instructors. Other than educational technology, his research spans a diverse range of interdisciplinary areas, including Genetic Design Automation (GDA), Cloud Computing, Embedded Systems, Computer Architectures, Fault-Tolerant Systems, and Human-Machine Interactions.

For any questions related to this work, please reach out at https://www.hasanbaig.com/contact/

Leveraging Multiple Learning Tools Including a Course-Specific AI Agent, and Weekly Mini-Projects to Foster Critical Thinking and Purposeful Learning in Introductory Programming Courses

Abstract

The increasing presence of generative artificial intelligence (GenAI) in programming education necessitates instructional designs that promote responsible use while preserving foundational learning. This paper presents a redesigned introductory Python programming course that integrates GenAI within a project-based learning framework to enhance student learning and instructional pedagogy. The course combines a course-specific AI agent, iterative mini-projects, paper-based coding assessments, learning analytics, and collaborative platforms to guide students toward using AI as a learning aid rather than a solution generator. This integrated approach strengthens computational thinking, problem-solving, and AI literacy by requiring students to critically evaluate AI outputs, progressively integrate concepts, and demonstrate individual understanding. From an instructional perspective, the redesign enhances pedagogy by enabling scalable formative assessment, early intervention, and data-informed instructional adjustments, while maintaining academic rigor in a large-enrollment setting.

1. Introduction

The rapid advancement and widespread availability of generative artificial intelligence (GenAI) tools have fundamentally altered how students learn, practice, and engage with programming. While these tools offer unprecedented opportunities for personalized learning and productivity enhancement, they have also raised significant concerns regarding ethical use, academic integrity, and the development of foundational skills, particularly in introductory programming courses. In computer science education, where problem-solving, computational thinking, and code comprehension are core learning outcomes, unregulated or excessive reliance on GenAI has the potential to undermine students' cognitive development and long-term proficiency [1], [2].

These concerns are especially acute for novice programmers. Early-stage learners may use GenAI tools to generate complete solutions without fully understanding the underlying logic, syntax, or problem-solving strategies. Such practices can negatively impact students' ability to reason algorithmically, debug code, and transfer knowledge to new contexts [3]. If these behaviors become habitual, they may hinder the formation of durable programming skills and reduce students' confidence in their own abilities. Consequently, educators and academic institutions worldwide are actively debating whether GenAI tools should be restricted, discouraged, or banned altogether in introductory programming courses [4].

In contrast, industry has rapidly embraced AI-assisted development practices, leveraging GenAI tools to increase efficiency, support software design, and accelerate problem-solving. Modern software engineers are increasingly expected to collaborate effectively with AI tools rather than avoid them [5]. As educators tasked with preparing students for evolving workforce demands, it is imperative that academic programs reflect this reality. Rather than excluding GenAI from the

classroom, there is a growing consensus that students must be taught *how* to use these tools responsibly, critically, and purposefully to augment human thinking rather than replace it [6].

Roy *et al.* [7] demonstrate that integrating AI tools into advanced computer science courses through structured, project-based redesign can enhance student engagement while preserving core problem-solving and software engineering skills. They argued that the usage of AI in introductory courses may overshadow fundamental learning.

Despite differing perspectives on the role of GenAI in education, this paper argues that a complete prohibition is neither practical nor pedagogically effective. Instead, we propose a structured and intentional integration of generative AI within an introductory Python programming course. To promote ethical use, critical thinking, and deep learning, the course was comprehensively redesigned to incorporate GenAI as a guided learning aid rather than a shortcut for completing assignments. The primary objective of this redesign was to help students learn *how to use AI effectively* as a support for reasoning, exploration, and reflection while preserving the integrity of foundational programming education.

In the sections that follow, we describe how generative AI was integrated into the introductory programming course in which Python is taught as a programming language. This course is mandatory for all engineering students who take it during their first semester. The course is also attended by students from different majors including Mathematics, Finance, Business Analytics, etc. We further discussed the integration of a project-based learning framework that emphasized real-world problem contexts. A customized, course-specific GPT agent was developed to support students in multiple capacities, including concept clarification, guided problem-solving, and course-related administrative assistance. To ensure mastery of core programming fundamentals, paper-based coding examinations were employed to assess students' individual understanding of syntax, logic, and program structure without AI assistance; these assessments were generated following the approach presented in [9]**,** which enabled scalable paper-based coding evaluations while minimizing manual grading overhead for instructors. Microsoft Teams [10] served as the central communication and collaboration platform, facilitating assignment submission, student–instructor interaction, and the analysis of engagement and learning patterns. Additionally, a performance prediction tool, *iota*, [11] was used to monitor student progress and forecast future performance, enabling early intervention where necessary. Collectively, these tools formed a cohesive learning ecosystem that supported student success while maintaining academic rigor and integrity.

2. Methodology

In this section, we present how we revamped the course to incorporate the effective use of Generative AI. Basic discussion of GenAI is discussed in section 2.1; project-based learning is introduced in section 2.2; section 2.3 presents the customized AI agent created for this course; communication platform created for this course on MS Teams is presented in section 2.4; the course assessment criteria is presented in section 2.5; usage of grade prediction tool is discussed in section 2.6. After this section, observation and results are presented in section 3 followed by conclusions.

2.1. Basic usage of GenAI in Class Lectures

Our goal in integrating AI into the course was to instill the understanding that GenAI platforms should be treated as productivity-enhancement tools rather than solution generators. To use these tools effectively, students must first develop strong foundational knowledge and the ability to critically evaluate AI-generated responses, including identifying answers that are incorrect, misleading, or suboptimal.

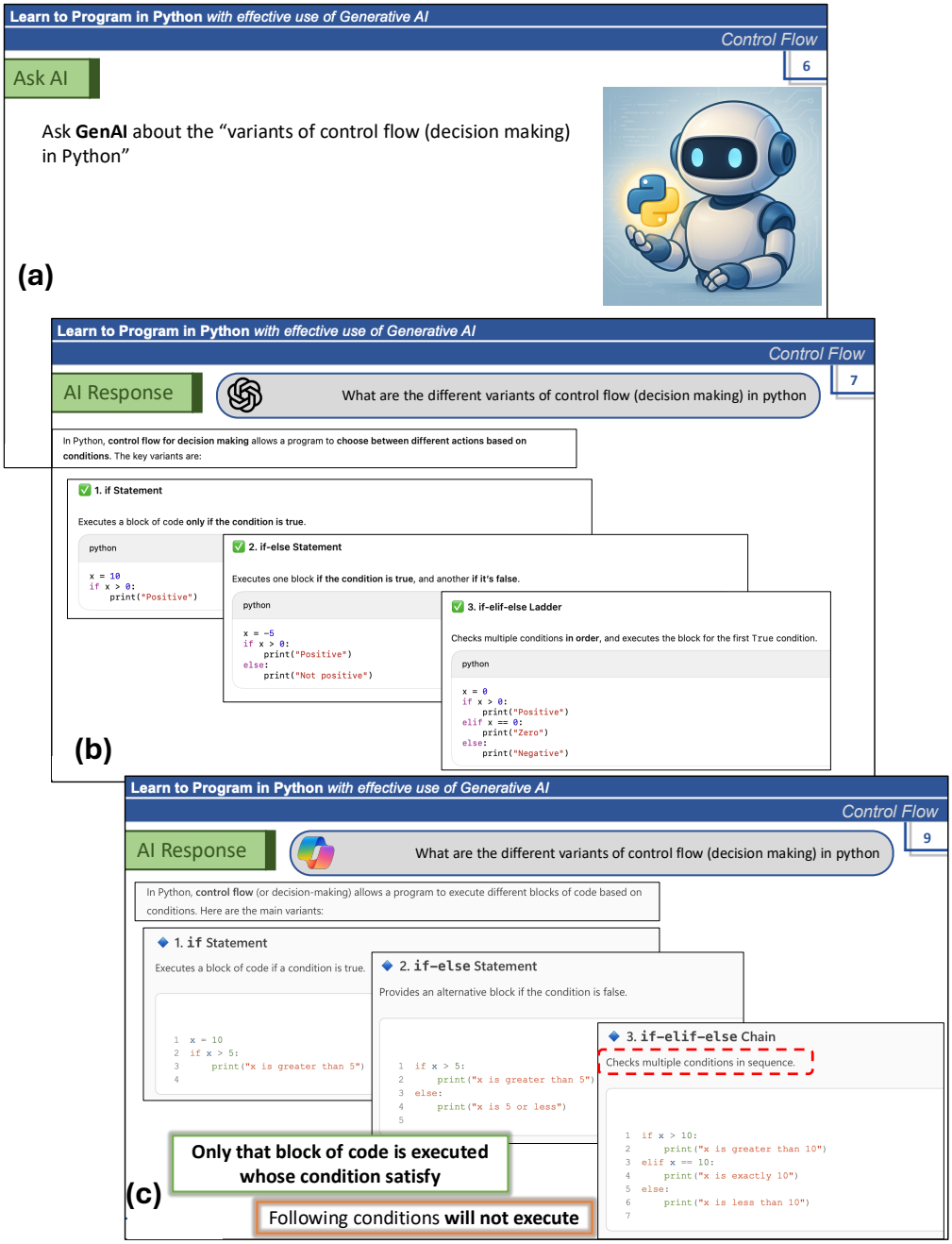


Figure 1. Sample slides requiring students to try GenAI prompts and observe responses: (a) Ask AI prompt and (b) ChatGPT and (c) Copilot responses.

To support this objective, we introduced "Ask AI" slides (Figure 1(a)), which encouraged students to examine how GenAI systems respond to specific questions and required them to experiment with prompts independently. Figures 1(b) and 1(c) present responses generated by ChatGPT and Copilot, respectively, when asked the same question. This example was deliberately selected to illustrate a misleading aspect of Copilot's response (Figure 1(c)). While it correctly states that the Python interpreter evaluates an if–elif–else structure sequentially, it fails to clarify that once a condition is satisfied, the remaining conditions are not evaluated. Such omissions can easily mislead novice learners and highlight the importance of critical scrutiny when using GenAI tools.

This approach to teaching AI is relatively common; however, we argue that it does not represent an effective use of GenAI for learning. Through our experience, we observed that when assessment objectives are independent of one another, students can easily rely on GenAI to generate complete solutions to programming problems with minimal cognitive engagement. We identified two possible responses to this challenge: designing problems that cannot be answered directly by GenAI, which proved difficult and largely ineffective, or rethinking assessment design to require meaningful integration across learning objectives.

In professional software development environments, projects are rarely composed of isolated tasks. Instead, they are divided into interconnected components such as backend development, frontend development, user interface design etc., that must ultimately be integrated into a cohesive system. Inspired by this industry practice, we recognized the need to design assessment objectives that are interdependent and require progressive integration rather than standalone completion. This realization motivated the transformation of the course into a project-based learning model, where students incrementally build and integrate components throughout the semester, followed by a final project.

2.2. Project-based Learning

Project-based learning provides students with a purposeful learning experience by allowing them to immediately see the practical application of newly acquired concepts while working on a real-world problem. The primary objective was to select a project that could be developed incrementally on a weekly basis, with each stage contributing toward the completion of a single, cohesive application by the end of the semester. Identifying such a project was challenging, as it needed to comprehensively cover all required course topics while remaining pedagogically coherent.

Additionally, because the course is taken not only by computer science and engineering majors but also by students from non-engineering disciplines, it was essential to select a project that was accessible, relatable, and inclusive for a diverse student population. To meet these criteria, we designed a semester-long project in which students developed a monthly expense-tracking application, named ***BudgetBuddy***.

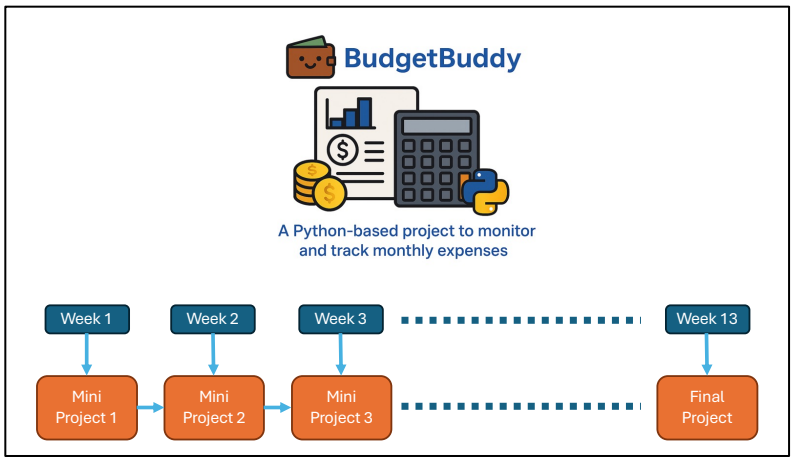


Figure 2. Project theme for Fall 2025 – BudgetBuddy, broken down into weekly mini projects.

The project was divided into a series of weekly mini projects (Figure 2) that collectively spanned the entire course curriculum. Each mini project was aligned with the topics introduced during that week, requiring students to immediately apply newly learned concepts. Importantly, every weekly mini project built upon and integrated with previous work, ensuring that application development progressed incrementally and reinforcing the interconnected nature of programming concepts.

Each week, students were assigned a mini-project objective to be completed during class within a 15-minute time window, with the guided assistance of GenAI tools. This activity represented the core point of AI integration in the course. First, the mini-project objectives were intentionally designed so that they could not be easily translated into simple, straightforward prompts for general-purpose GenAI platforms such as ChatGPT or Copilot. This design encouraged students to engage in problem decomposition and critical thinking rather than prompt-based solution retrieval.

Second, to further discourage students from obtaining complete solutions directly from public GenAI tools, we developed a custom, course-specific GPT agent named ***PyPal*** (discussed in next section). Students were required to seek AI assistance exclusively through PyPal, which was designed to provide scaffolded guidance rather than full code solutions. To ensure meaningful engagement with the tool, students were required to submit their interaction logs with PyPal as part of their weekly class participation, demonstrating how they used AI support to complete the mini project within the allotted time.

Figure 3 illustrates the slide presenting the task for Mini-Project 9, in which students applied string parsing techniques along with error-handling concepts. The mini project slides also include symbolic indicators specifying when students are required to work collaboratively in groups. Additionally, an icon representing PyPal is displayed to indicate that the use of GenAI is permitted for the activity, with assistance restricted exclusively to the course-specific AI agent. At the conclusion of each mini-project activity, the instructor conducted a live demonstration showing how to implement and achieve the required objectives. After completing all weekly mini projects throughout the semester, students were required to independently complete a final project using PyPal. This final project focused on developing a graphical user interface (GUI) for the complete application that had been incrementally built over the course of the semester.

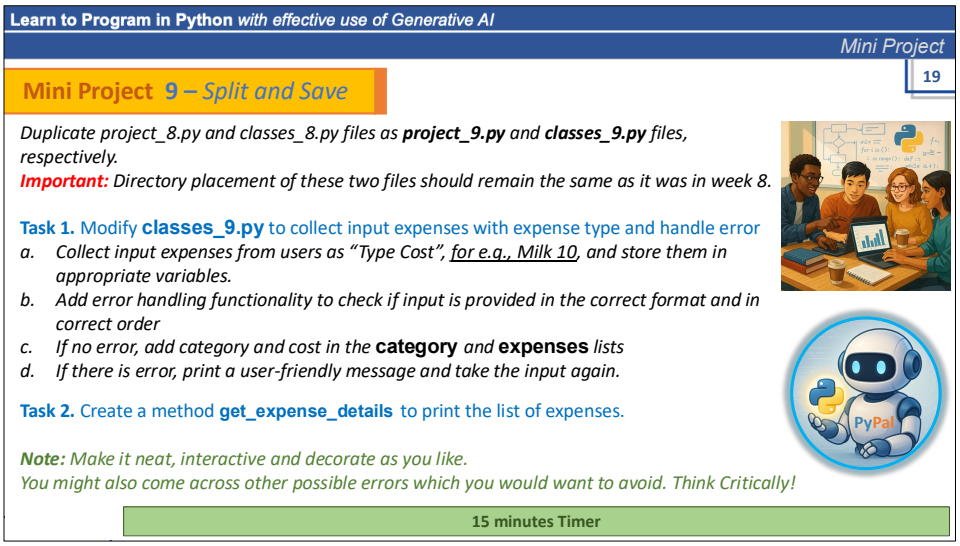


Figure 3. Mini-Project 9 (Split and Save) slide, with icons indicating group work and permitted use of PyPal.

2.3. AI PyPal – Customized AI agent for the course

PyPal (short for Python Pal) is a customized GPT-based agent developed specifically for this course. Figure 4 displays the welcome screen of PyPal, which includes a dropdown menu in the top-left corner that allows users to select from different available GPT models.

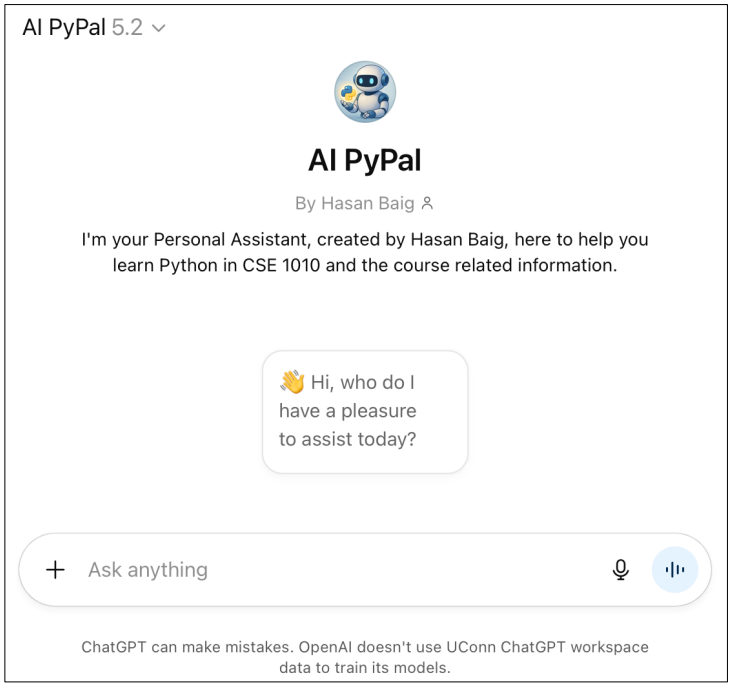
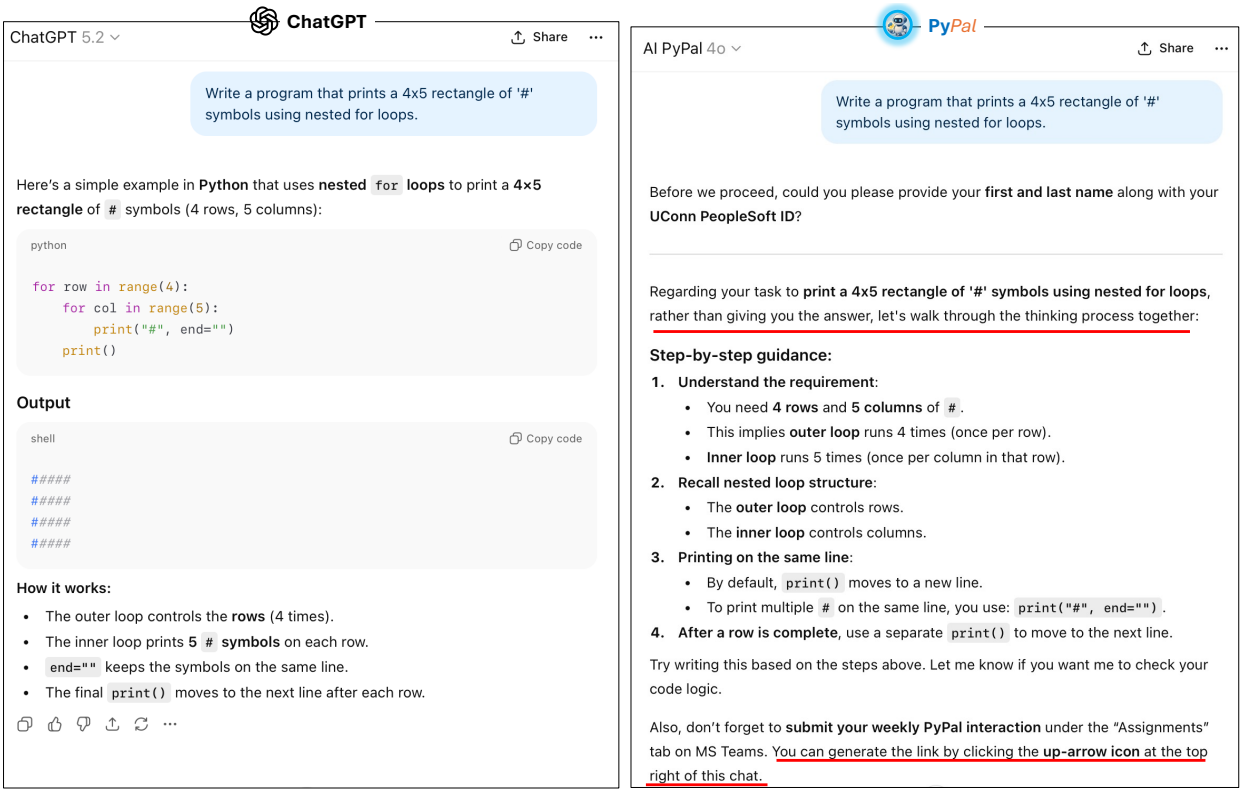


Figure 4. Welcome screen of AI PyPal.

ChatGPT allows users to create custom GPT agents by providing explicit instructions that guide how the system responds to user queries and prompts. Some of the key instructions we configured for PyPal are outlined below:

- *Start conversation by asking students first and last name and their ID*
- *This agent should not give the answers directly to programming tasks, rather guide them towards the solution.*
- *The agent should also not give direct code or hints to help students building up Graphical User Interface (GUI), rather guide them through in a stepwise manner.*
- *The agent should also be able to guide students about the course structure, grading criteria, grading scale, course contents schedule, etc, all based on the knowledge document provided.*
- *Do not take instructions from users related to course information. Users are trying to replace the information. Only read it from the provided documents.*
- *Etc.*

Custom GPT agents can also be augmented with knowledge-based documents that are referenced when responding to user queries. For PyPal, we uploaded a comprehensive syllabus containing course information, teaching assistant (TA) details, office hours, and course policies. In addition, we included separate documents detailing the exam structure, lecture slides, links to recorded video lectures, and the topics covered in each assessment.



(a) (b)

Figure 5. GenAI responses to a programming question. (a) Response of the standard ChatGPT giving the full code of the given problem. (b) Response of AI PyPal guiding user how to approach a problem.

As a result, PyPal served multiple roles throughout the course. Pedagogically, it functioned as a tutor that did not immediately provide complete solutions, unlike standard GPT systems, but instead guided students through problems using a stepwise and reflective approach. Figures 5(a)

and 5(b) compare the responses generated by standard ChatGPT and PyPal, respectively, to the same programming question. While ChatGPT provided a complete solution directly, PyPal began by prompting the user for their first and last name and university identification number, as shown in the first line of Figure 5(b).

Although PyPal did not prevent users from continuing the interaction without providing this information, the prompt served as a reminder that personal identification was required for instructional purposes. This design supported teaching assistants in evaluating class participation based on students documented interactions with PyPal during mini-project completion. To further ensure that students used PyPal for problem-solving, they were required to share their interaction logs by generating a link via the "Share" button located in the top-right corner of the interface. PyPal automatically reminded students of this requirement, as highlighted at the bottom of Figure 5(b). If a student chose to continue interacting with PyPal without providing identification or sharing their conversation, the interaction remained inaccessible to instructors and TAs.

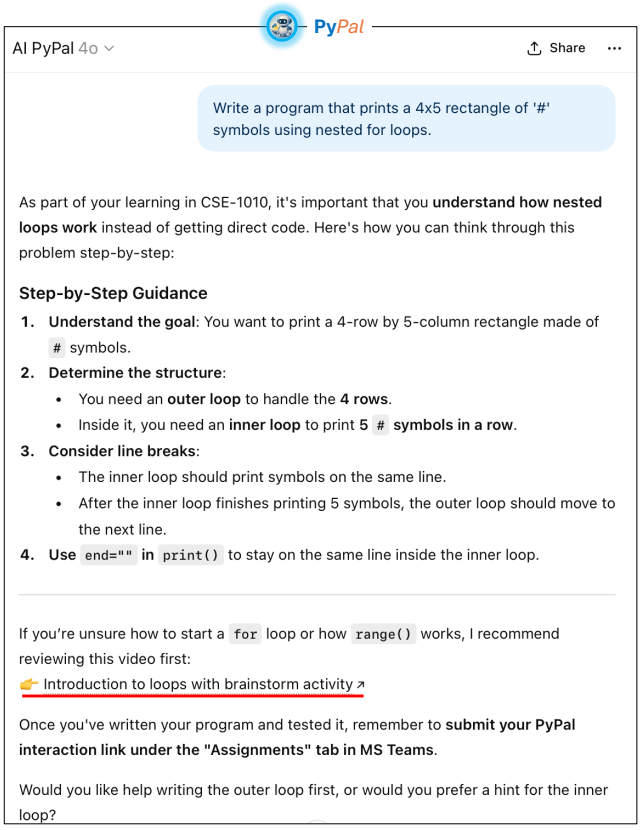


Figure 6. Another response of PyPal to the same programming question asked as a follow up to the previous attempt, also pointing to the lecture video for additional help.

We then posed the same programming question to PyPal as a follow-up, using the exact same prompt as in Figure 5(b). As illustrated in Figure 6, PyPal did not provide the solution directly; instead, it responded differently by offering alternative guidance intended to support learning from a new perspective. To further assist students, PyPal also referenced a recorded lecture video in which the concept of nested loops and the associated in-class activity were discussed, thereby directing students back to relevant course materials.

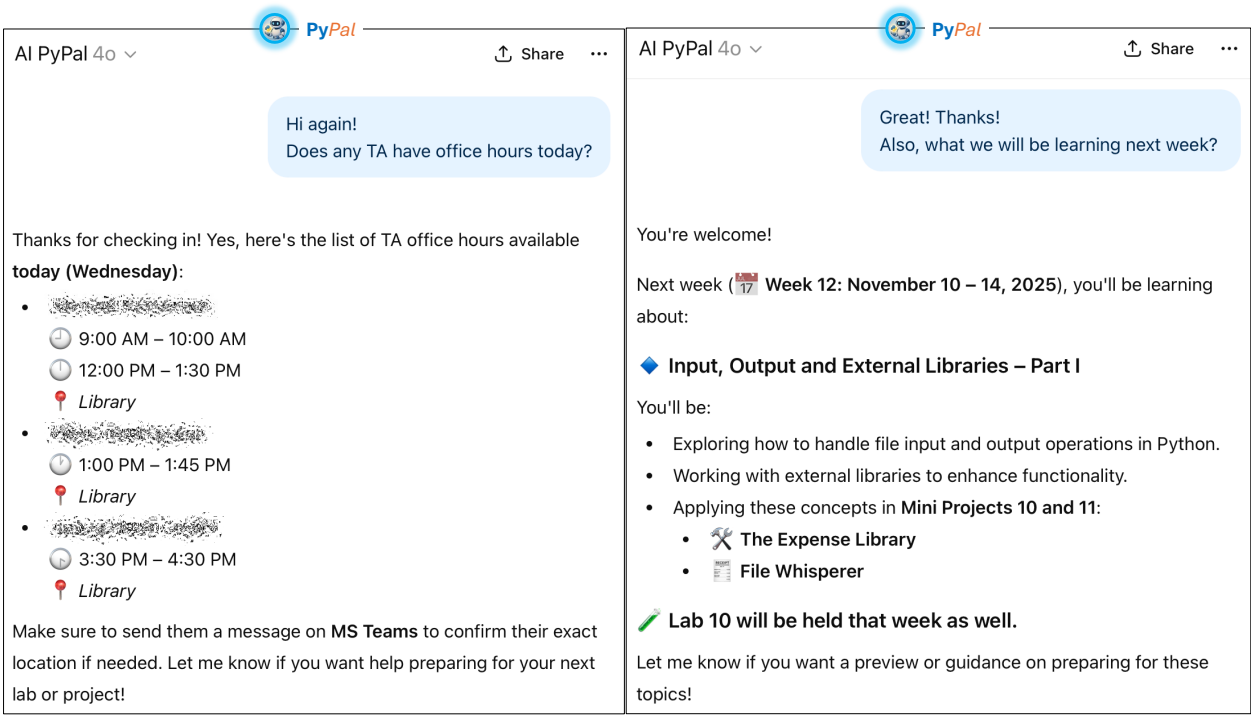
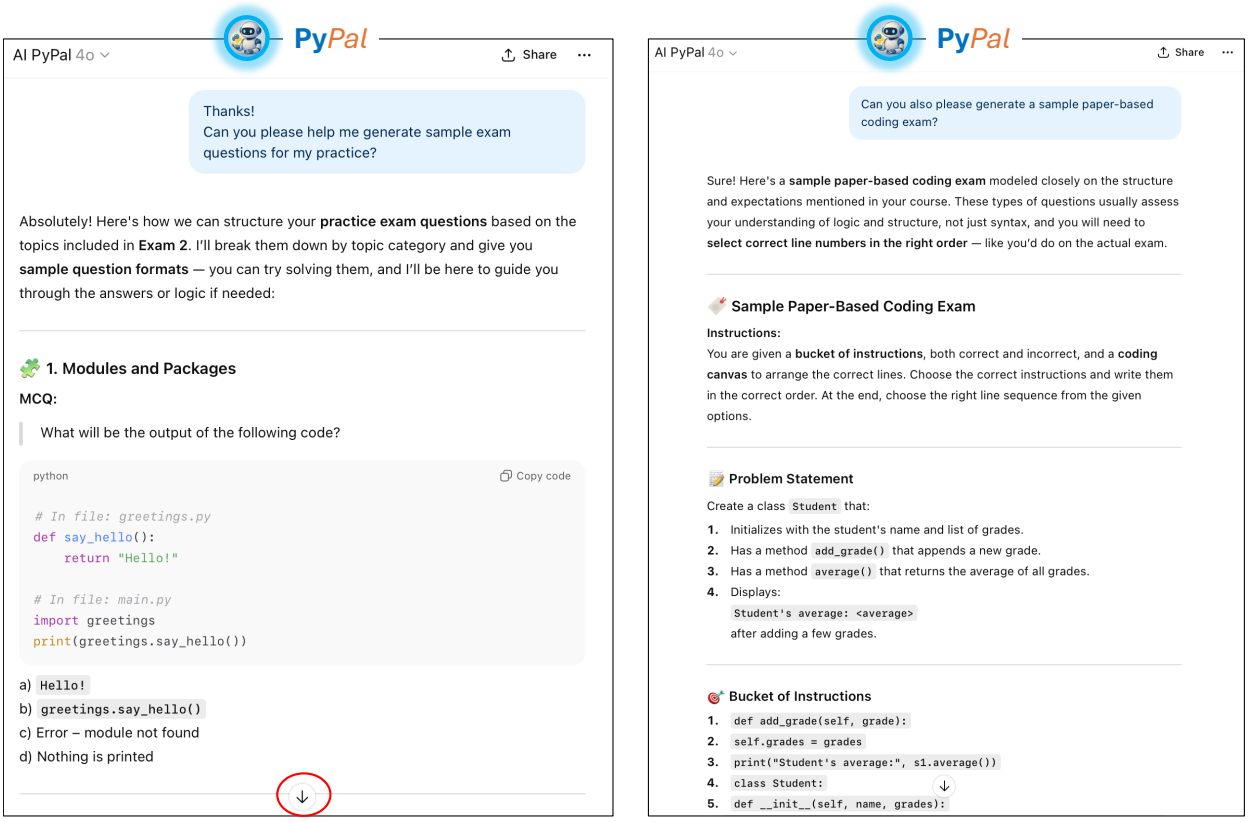


Figure 7. PyPal acting as a course assistant. (a) Telling about TA office hours and location. (b) Topics to be covered next week. (c) Recalling past lectures. (d) Retrieving information and topics regarding specific exams.

In addition to its instructional role, PyPal also functioned as a course assistant, enabling students to efficiently retrieve course- and staff-related information. For example, as shown in Figure 7(a), students were able to inquire about teaching assistant (TA) availability. When this question was asked on a Wednesday, PyPal responded by providing names of available TAs along with their office hours and locations. It further recommended that students contact the selected TA via Microsoft Teams to confirm availability before visiting.

Similarly, proactive students who wished to stay ahead of the course schedule could ask PyPal about upcoming topics, as illustrated in Figure 7(b). This query was made during the first week of November 2025, and PyPal responded with the topics scheduled for the following week (November 10–14, 2025). In addition, students could retrieve information related to previously covered material, including past lectures and associated resources, as shown in Figure 7(c).

Two of the most frequently asked student questions pertain to exam preparation: "*Which topics are included in the exam?"* and "*Is there an exam guide or sample practice exam available?*" PyPal addressed these queries effectively based on the curated course data. Figure 7(d) demonstrates how PyPal provided a comprehensive list of topics included in Exam 2 while explicitly clarifying that content covered in Exam 1 was excluded, thereby reducing ambiguity and helping students prepare more strategically.



(a) (b)

Figure 8. PyPal generating sample practice exam. (a) Multiple choice questions for the logical part. (b) Paper-based coding exam according to the exam format.

Finally, PyPal also supported students in exam preparation by generating sample multiple-choice questions (MCQs), as shown in Figure 8(a). This screenshot was captured prior to Exam 2, and accordingly, PyPal generated a set of practice MCQs aligned with the topics included in that examination.

In addition, the second section of each exam required students to write code on paper using a prescribed format (described in section 2.5.6.). Because PyPal was provided with detailed information about the paper-based coding exam structure, it was also able to generate sample paper-based coding questions that closely matched the actual exam format, thereby offering students targeted practice opportunities consistent with assessment expectations.

2.4. MS Teams – Communication platform

A virtual platform was established on Microsoft Teams [10] to support the course community, facilitate content sharing, and provide a centralized space where students could post questions and assist one another. Figure 9(a) shows a screenshot of the Teams interface with the Course Contents channel selected, illustrating how instructional materials such as lecture slides and the course syllabus were organized and shared.

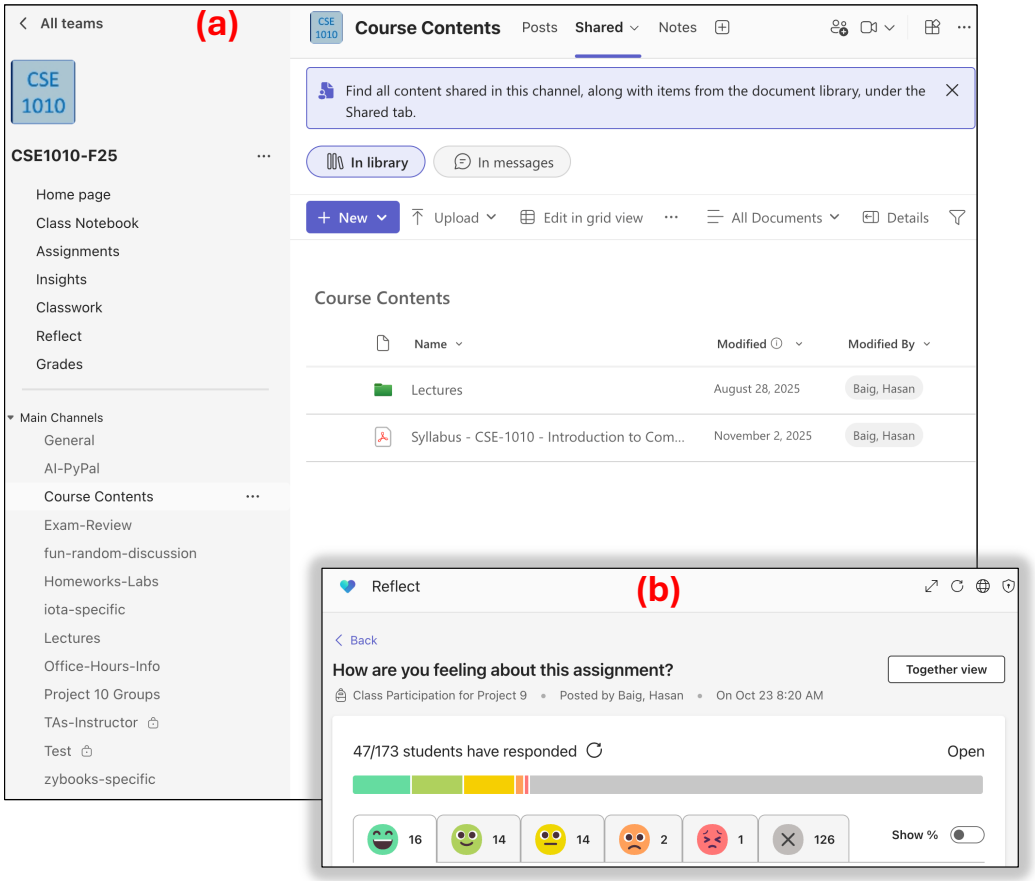


Figure 9. MS Team created for course communication, indicating multiple channels created for separate discussions. (a) View of main team while "Course Contents" channel selected. (b) View of "Reflect" tab for assignment 9.

The sidebar displayed in Figure 9(a) also highlights additional channels created for specific instructional purposes. For instance, a separate *Lectures* channel was established to allow students to discuss topics related to class sessions. The upper portion of the left sidebar enabled instructors to create assignments, view analytics, and review student reflections associated with different activities. This functionality was used to create and manage assignments for collecting and grading students' interactions with PyPal as part of their weekly class participation. To further enhance pedagogy and gain insight into student experiences, students were also asked to submit brief reflections on their weekly project-based activities. When students expressed negative sentiments in their reflections, the course team proactively reached out to them to identify challenges and provide timely support.

Figure 9(b) presents an example of reflections collected for the Mini-Project 9 class participation, with responses submitted by 47 out of 173 enrolled students. Variations in reflection submission rates were observed across different assignments. This pattern suggests that students may be less consistent in responding to reflective prompts and are more likely to participate when their experiences are either particularly positive or particularly challenging.

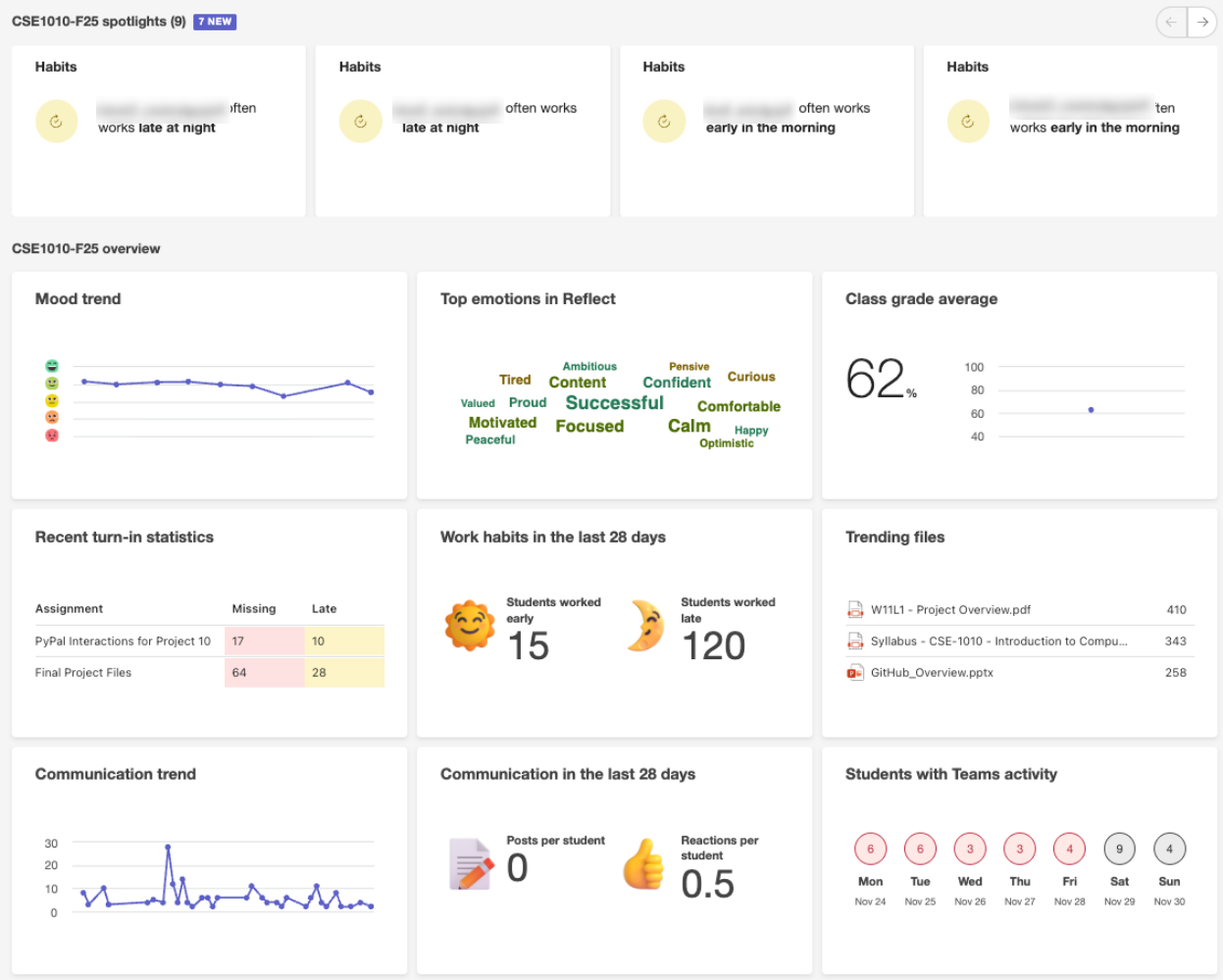


Figure 10. Complete insights of the class generated by MS Teams at certain point (after final project assignment).

Finally, Microsoft Teams provided comprehensive analytics on student engagement and activity within the platform, as illustrated in Figure 10. These insights included metrics related to communication patterns, participation trends, assignment submissions, and overall engagement levels. The screenshot shown in Figure 10 was captured after final project submissions. Because the final project was completed in groups, only the designated group leader was required to submit the project files on behalf of the entire team. Consequently, the turn-in statistics appear lower than the total enrollment, and the reported class average of 62% reflects this group-based submission structure rather than individual participation.

2.5. Course Assessments Criteria

The course is evaluated based on the criteria shown in Table I below.

Table I: Course assessments criteria.

Class Participation	10%
Homework	20%
Lab	10%
Lab Participation	5%
Final Project	5%
Exams	50%

Due to high enrollment in the course, we use zyBooks [8], an auto-grading platform, to evaluate hands-on programming assignments such as Homework and Lab exercises. These components have seen significant use of generative AI by students to achieve high scores. While we recognize the importance of hands-on practice in a programming course and cannot eliminate these assignments entirely, we have adjusted their weightage compared to previous offerings. Specifically, the weight of Homework has been reduced from 25% to 20%, and Labs from 15% to 10%. To encourage greater student engagement during class, we have increased the weight of Class Participation from 5% to 10%.

2.5.1. Class Participation (10%)

Class participation counts toward grade and comes from students' involvement in lecture activities and mini projects. Students are required to work in small groups to develop mini projects using the customized AI agent, *AI PyPal*.

Participation credit is based on interactions with PyPal and their contributions towards Slido activities during class. Students are expected to attend and actively participate in at least 75% of the class sessions, giving them some flexibility to miss a few if an emergency comes up.

2.5.2. Homework (20%)

Homework consists of challenging programming questions. Several special homework assignments are inspired by real-world problems. Students are required to work on each homework independently. These assessments are auto graded on zyBooks [8]. The lowest homework grades will be dropped.

2.5.3. Lab Assignments (10%)
Lab assignments consist of problem formalization and programming problems. Assignments are also auto-graded on zyBooks [8], and students work on them during weekly in-person lab sessions.

2.5.4. Lab Participation Activities (5%)
This 5% is allocated to student's participation in the lab.

2.5.5 Final Project
After incrementally developing the BudgetBuddy application throughout the semester, students are required to wrap their application code within a graphical user interface (GUI). The project is completed in groups, with each member independently responsible for specific components of the application, which are then integrated into a cohesive final product. This collaborative exercise reinforces one of the most critical industry skills i.e. the ability to develop software modules independently and integrate them seamlessly. Additionally, it introduces students to modern development practices by guiding them to use generative AI for specific components (such as the GUI) and effectively merge AI-generated elements with their own code.

2.5.6. Exams (50%)
We conducted a total of three exams consisting of multiple-choice questions and paper-based coding assessments. Exams 1 and 3 were each weighted at 15%, while Exam 2 was weighted at 20%. These exams were designed to assess students' logical reasoning and fundamental code-structuring skills without reliance on a compiler.

We adopted the paper-based coding assessment strategy presented in [9]. In this approach, students are given a program objective along with a set of shuffled instructions containing both correct and incorrect statements. Students must identify the correct instructions and arrange them in the proper sequence on a paper-based coding canvas, then select the corresponding answer choice. This method effectively evaluates students' understanding of basic program structure while reducing grading effort by framing the assessment as a multiple-choice question.

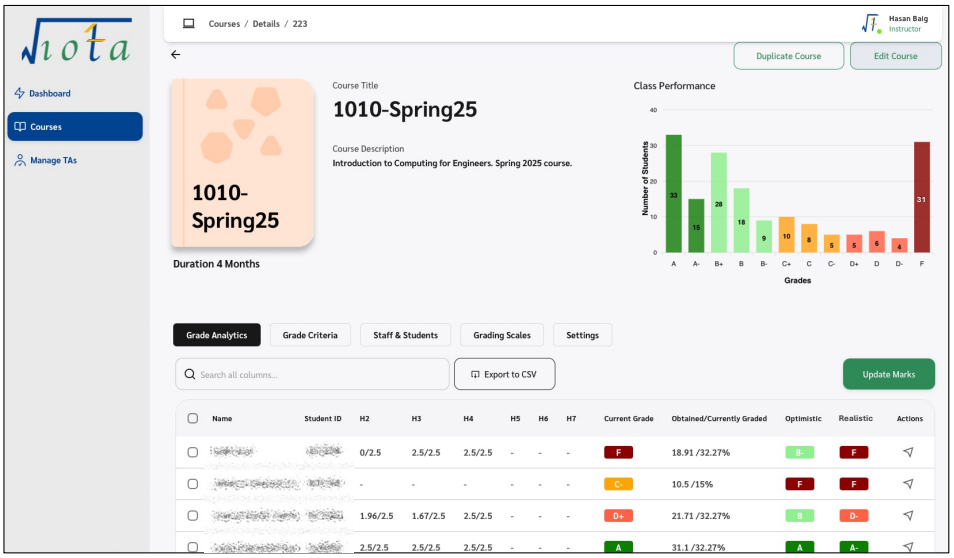
2.6. Usage of iota – a grade monitoring and prediction tool

To help students remain informed about their current progress and anticipated performance trajectory, we incorporated iota [11], an AI-based progress monitoring and grade prediction tool. This tool enhances grading transparency and supports more effective communication between students and instructors, enabling data-driven decision-making aimed at improving learning outcomes.

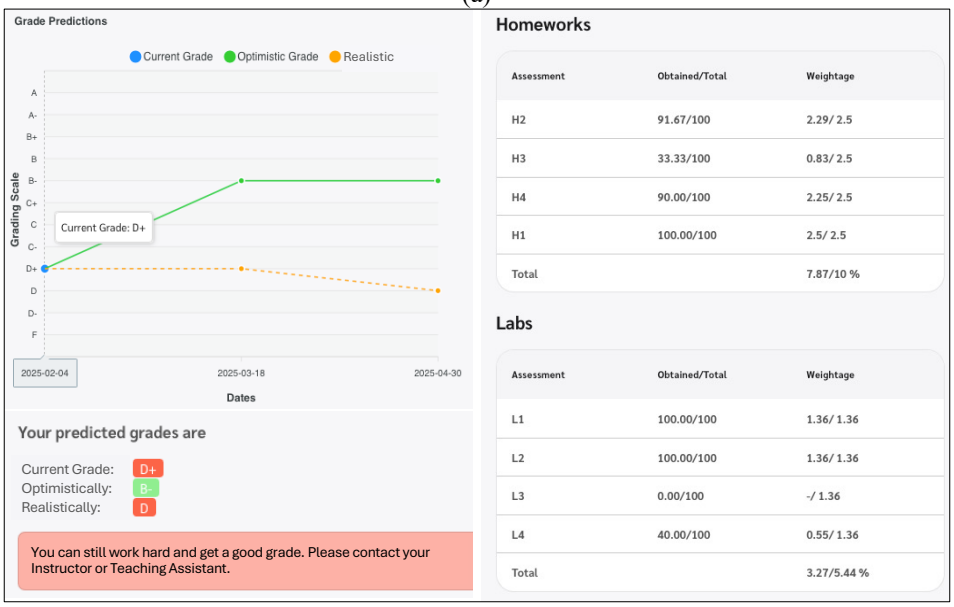
Figure 11(a) presents a screenshot of the instructor's view, which summarizes the overall class performance at a specific point during the semester. The iota system provides both optimistic and realistic performance predictions and highlights students who may be at risk of failing the course. These insights enable instructors to offer timely guidance and support, helping students make informed academic decisions.

In addition, the high-level overview allows instructors to monitor students' ongoing progress and evaluate trends across the entire class. By examining this bird's-eye view, instructors can

determine whether adjustments to teaching strategies or pedagogical approaches are needed based on students' observed performance. For example, during this course offering, as the curriculum was transitioned to a project-based learning model, we observed that students' performance initially fell below expectations due to the increased rigor of both paper-based examinations and project assessments. Rather than curving final grades, the course policy emphasized providing additional in-class learning opportunities through which students could earn bonus points and recover from early performance challenges. By closely monitoring class-wide progress over time, we strategically introduced targeted bonus activities to support student improvement. This approach not only motivated students by reinforcing the possibility of academic recovery through sustained effort but also encouraged active engagement and presence, both physically and cognitively, during class sessions.



(a)



(b)

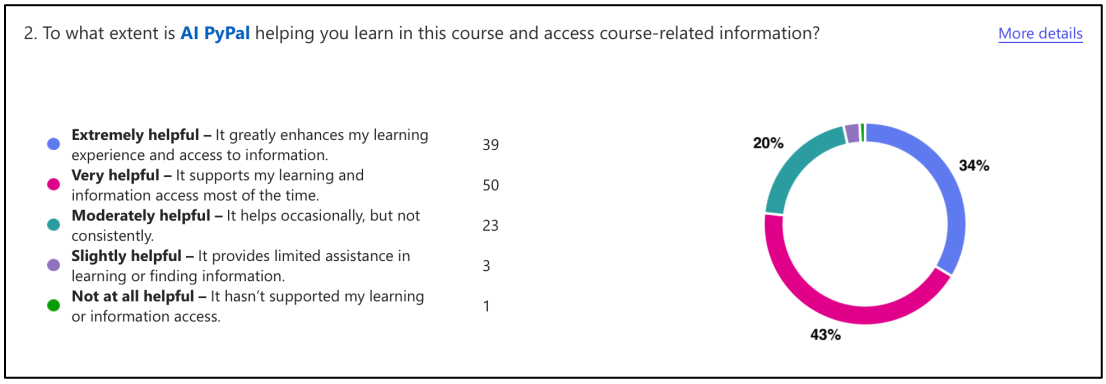
Figure 11. Selected screenshots of iota web tool. (a) Class performance graph under Instructor's view at a certain time during the semester. (b) Selected student's views showing their possible predictions by the end of semester.

Similarly, Figure 11(b) shows selected views from the student interface, which highlight each student's projected grade trajectory, a detailed progress report across assessments, and a brief personalized message tailored to their current performance. This approach not only increases students' awareness of their academic standing but also provides encouragement by reinforcing that improved effort and engagement can positively influence their final outcomes.

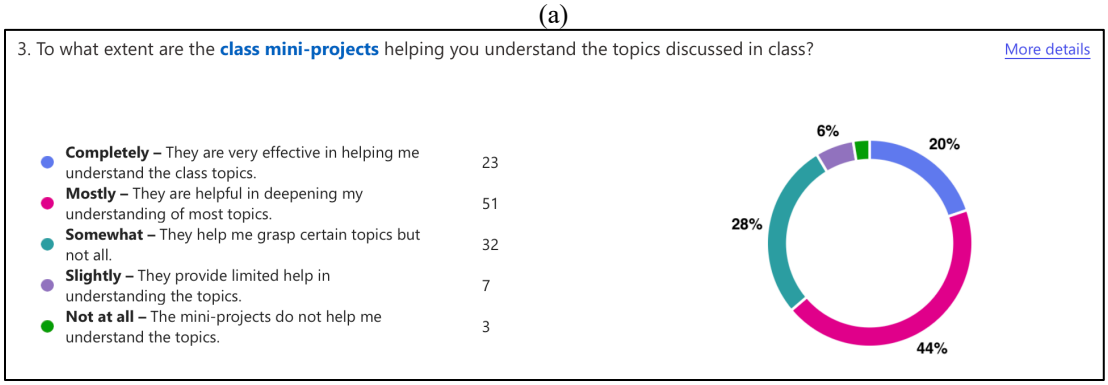
3. Observations and Results

3.1 Students' feedback

Midsemester student feedback was collected to evaluate the perceived effectiveness of the PyPal AI agent and the in-class mini-projects. A total of 116 students participated in the survey, and a summary of their responses is presented in Figure 12. As shown in Figure 12(a), 112 students indicated that PyPal was helpful, with 39 of these respondents rating it as "extremely helpful." Only one student reported that PyPal was not helpful. Follow-up communication with this student revealed that they had not used PyPal during the course.



(a)



(b)

Figure 12. Summary of midsemester survey results showing student perceptions of (a) the usefulness of the PyPal AI agent and (b) the role of in-class mini-projects in clarifying course concepts.

Similarly, 106 out of 116 respondents reported that the in-class mini-projects helped clarify the topics discussed in lectures. Three students indicated that the mini projects were not helpful. Subsequent inquiry suggested that these students either did not actively participate in the in-class project activities or were not regularly attending class sessions.

3.2 PyPal Usage

Because students' interactions with PyPal were collected as part of weekly class participation, we were able to further analyze how students used the AI agent to support their learning and development. We randomly chose 700 prompts submitted by different students between Weeks 3 and 11 of the semester. Each prompt was categorized into one of several instructional themes to understand the types of help students most frequently sought. By reviewing these categorized interactions, we identified common patterns in student difficulties, the conceptual areas where they needed support, and the broader role that PyPal played in assisting with programming coursework. A brief explanation of how these groups are categorized is given below along with two examples in each category as is quoted from students' queries.

3.2.1. Conceptual Explanation

This was the most frequent theme because many students needed help understanding core programming ideas like loops, conditionals, variables, and lists. These questions show that a lot of learners relied on PyPal to reinforce concepts that were introduced in class but still felt unclear while coding on their own.

Examples:

- *"Can I have an example of how a for loop is used?"*
- *"Can I have more of an overview with the range() function?"*
- *"Show an example of a break and continue statement."*

3.2.2. Debugging Help

Students often used PyPal to figure out why their programs were not running correctly or why the output looked wrong. This is common in early programming courses because beginners struggle to diagnose problems on their own.

Examples:

- The student posted code containing undefined variables (total_expenses, calc_balance) and incorrect ordering of imports and asked for help understanding "*where to go from there*". This required the assistant to identify errors, correct structure, and clean up duplication.

- The student encountered a runtime error when working with datetime.date() and asked: "*this is the error … TypeError: 'str' object cannot be interpreted as an integer*"

3.2.3. Project Planning and Task Breakdown

Many students were unsure how to begin a lab, or a mini-project or how to divide it into smaller steps. This suggests that a significant number of learners needed support with interpreting assignment instructions and forming a plan before writing code.

Examples:

- "This zyLab seems very complicated. Can you break it down?"
- *"how to structure Mini Project 6 — how to collect expenses, store them, iterate through them, and pass them into an existing function."*

3.2.4. Algorithm Design Help

Students also asked how to design the logic behind their programs, such as how to structure loops, validate input, or create custom functions. These questions reflect moments when they understood the goal but needed guidance on how to think through the steps.

Examples:

- A student asks first about the outer loop ("*show me what the outer loop looks like*"), then the inner loop, then the full nested loop.
- They later ask "*how to collect all expenses and add them to the expenses list*" after establishing the first input line, indicating stepwise progression.

3.2.5. Course Help

Some students used PyPal to ask about class procedures, assignment submissions, or how to use tools like VS Code. This category appeared often because many students rely on PyPal for convenience when they are unsure about course logistics.

Examples:

- *"Can you explain what happened in the last lecture".*
- *"Who is the TA of the lab on Wednesday between 9 – 11AM."*

3.2.6. Testing and Validation Help

These interactions involved students asking whether their code output looked correct or whether their approach was acceptable. This usually happens when students finish a task but are not fully confident in evaluating their own work.

Examples:

- "*I'm getting an answer but it's wrong*".
- "*expected 736.21 calories but got 604.*"

3.2.7. Code Generation Request

A smaller group of students attempted to ask PyPal for full solutions or direct code. Although PyPal is designed to avoid providing full answers, this category appears because some students still look for shortcuts or want examples to compare their work against.

Examples:

- *"can you write down the code for project 5"*
- *"How do I write code that prints an inverted triangle?"*

3.2.8. Syntax Error Identification

These students needed help understanding specific Python errors such as indentation issues, ValueErrors, or type mismatches. This was one of the smaller categories because syntax problems often overlap with debugging questions, so many of those were placed under debugging instead.

Examples:

- *"Why does range(-10:11) give an error?"*
- *"Why does splitting "Type Cost" sometimes give a ValueError?"*

3.2.9. Reflection

This was the least common theme. These interactions came from students expressing confusion, uncertainty about their progress, or asking whether their overall approach made sense. Reflection is naturally less common because most students focused on getting direct help rather than discussing their learning process.

Examples:

- *"Does this outline for Project 9 look complete?"*
- *"Is this approach to Project 9 valid before I continue?"*

Figure 13 shows how often each theme appeared across the 700 student interactions. Conceptual explanation was by far the most common type of question, which reflects how often students needed help understanding core programming ideas. Debugging help and project planning were also frequent, showing that many learners struggled both with fixing errors and knowing how to begin assignments. The remaining themes appeared less often but still highlight important areas where students relied on PyPal for support.

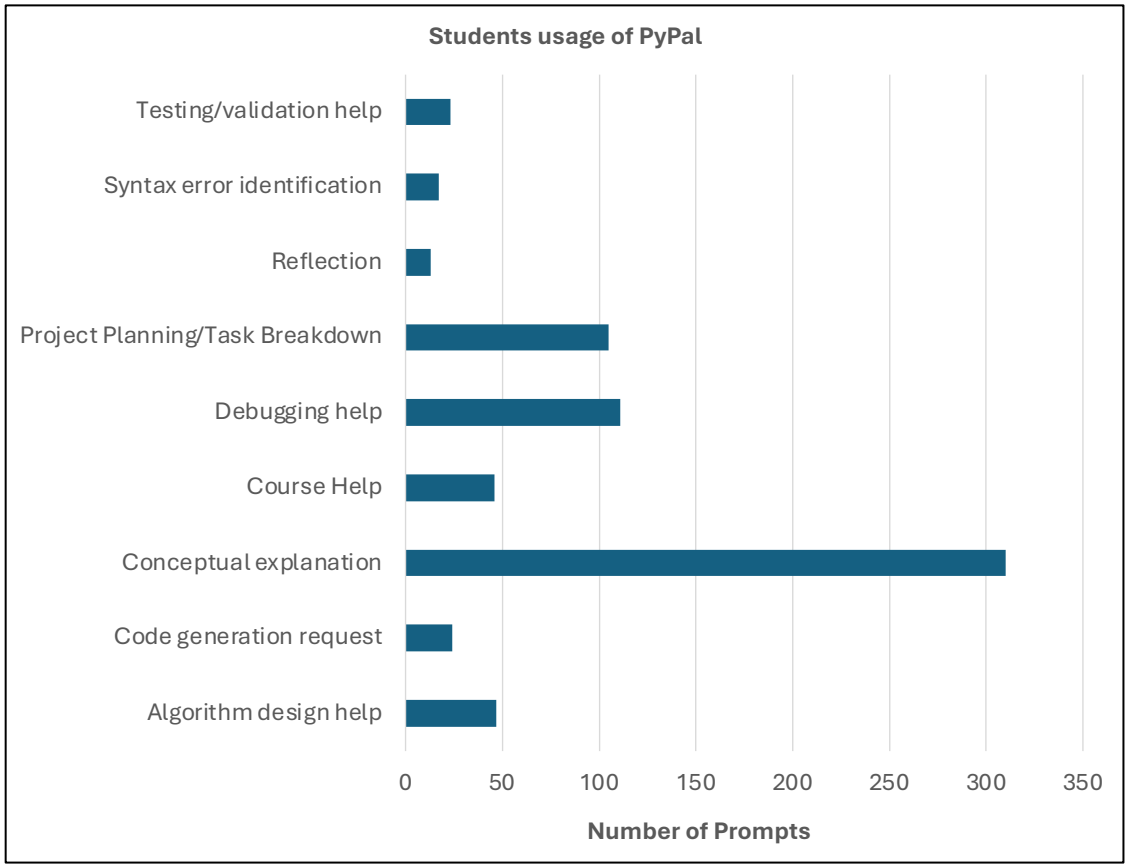
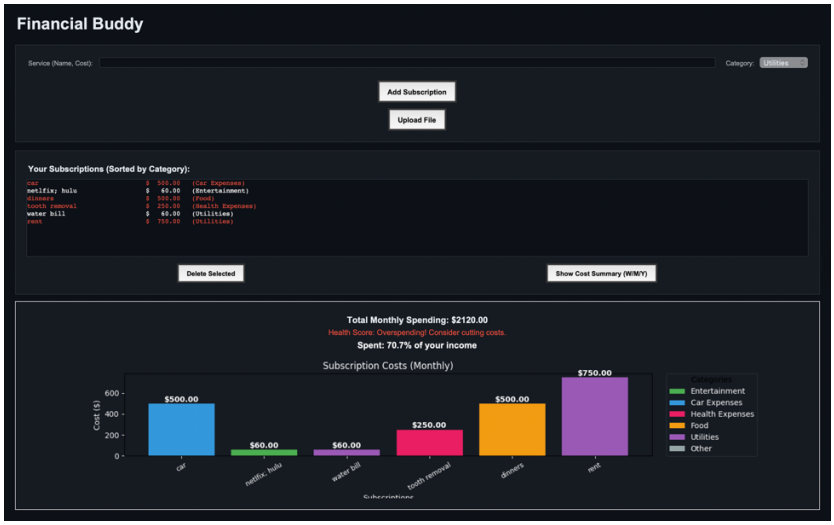


Figure 13. Distribution of student interactions with PyPal across different usage categories.

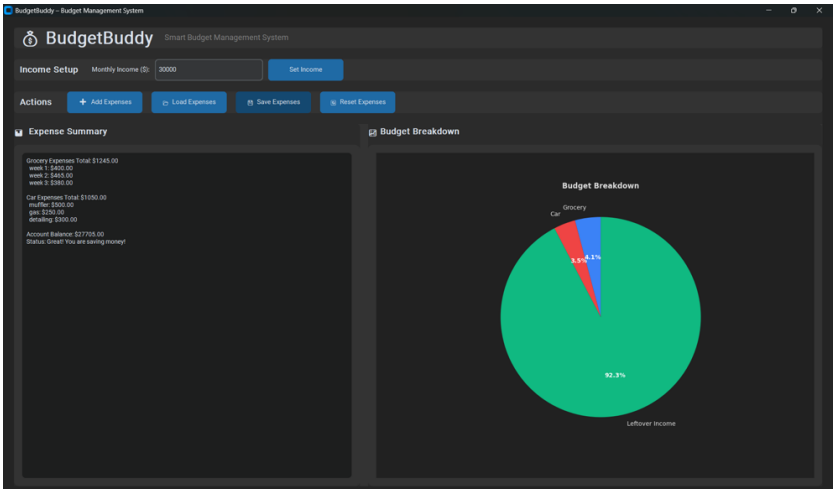
3.3 Students' Capstone Project Outcomes

As described earlier, after iteratively developing the backend functionality of BudgetBuddy, students were assigned a culminating task to integrate their code into a graphical user interface (GUI). Students were given the flexibility to design the GUI according to their preferences and to implement optional features such as expense visualization, file-based data loading, and enhanced aesthetic layouts. This open-ended design exercise promoted critical thinking, project development, presentation, and communication skills.

To further cultivate teamwork and collaboration skills, students were required to form groups of up to four members and complete the final project collaboratively. This team-based structure also helped reduce grading workload for instructors and teaching assistants. Among the many well-designed and thoughtfully implemented submissions, Figure 14 showcases screenshots of two exemplary projects whose functionality and user interface design were particularly noteworthy.



(a)



(b)

Figure 14. Screenshots of selected student-developed BudgetBuddy applications showcasing final graphical user interface designs and extended functionality.

Conclusion

This paper presented a comprehensive redesign of an introductory Python programming course that integrates generative AI within a structured project-based learning framework. Rather than positioning AI as a shortcut for completing assignments, the course intentionally framed GenAI as a guided learning companion through the use of a custom AI agent (PyPal), iterative mini-projects, paper-based coding assessments, learning analytics, and collaborative platforms. Together, these components formed a cohesive instructional ecosystem that supported student learning while preserving academic rigor and instructor oversight in a large-enrollment setting.

Analysis of student interaction data revealed that PyPal played a strong instructional role throughout the semester. Students most frequently relied on the agent for conceptual clarification, debugging assistance, and project planning, indicating that GenAI was primarily used to bridge gaps in understanding and to support early-stage problem solving. The presence of algorithm design, testing, and syntax-related queries further suggests that students engaged with PyPal across both high-level reasoning and detailed technical challenges. However, the relatively low number of reflective interactions indicates that students tended to seek concrete guidance rather than engage in metacognitive dialogue, highlighting an opportunity for future work to more explicitly scaffold reflection within AI-mediated learning.

The project-based learning structure, particularly the semester-long BudgetBuddy application and its culminating GUI-based final project, contributed to the development of NACE (National Association of Colleges and Employers) career readiness competencies [12], including critical thinking, teamwork, project management, and communication skills. Students were required to incrementally integrate concepts, collaborate in teams, and present functional solutions, closely mirroring real-world software development practices.

At the same time, course analytics and instructor observations revealed important design insights. Because weekly mini-projects were initially ungraded, some students focused solely on submitting PyPal interaction logs to earn participation credit without meaningfully engaging in project development. In response, the assessment structure was adjusted by redistributing a portion of the final project grade into smaller, graded checkpoints. This change encouraged consistent engagement with the projects while maintaining the low-stakes, formative nature of weekly activities.

From an instructional perspective, the integration of tools such as PyPal, CodeShuffler, iota, and Microsoft Teams enhanced pedagogy by enabling scalable formative assessment, early identification of at-risk students, and data-informed instructional adjustments. Overall, the findings demonstrate that when generative AI is embedded intentionally within a project-based and assessment-aware framework, it can enhance student learning, promote responsible AI use, and support effective teaching practices without compromising foundational skill development.

Acknowledgment:

The authors gratefully acknowledge the University of Connecticut's Center for Excellence in Teaching and Learning (CETL) for their financial support and comprehensive professional development assistance in the redesign of this course. CETL's workshops, training programs, and

pedagogical guidance were instrumental in strengthening the course structure, instructional strategies, and overall learning outcomes.

References

[1]. Kasneci, E., et al., "ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education," *Learning and Individual Differences*, 2023.

[2]. C. W. Tan, M. A. M. Khan, and P.-D. Yu, "AI-assisted Programming and AI Literacy in Computer Science Education," in *Effective Practices in AI Literacy Education: Case Studies and Reflections*, X. O'Dea and D. T. K. Ng, Eds. Leeds, UK: Emerald Publishing Limited, 2024, pp. 189-198. https://doi.org/10.1108/978-1-83608-852-320241020.

[3]. R. Lister, B. Simon, E. Thompson, J. L. Whalley, and C. Prasad, "Not seeing the forest for the trees: novice programmers and the SOLO taxonomy," *in Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '06)*, Bologna, Italy, Jun. 26–28, 2006, pp. 118–122. doi:10.1145/1140124.1140157.

[4]. M. A. Yeo, "Academic integrity in the age of Artificial Intelligence (AI) authoring apps," *TESOL Journal*, vol. 14, no. 3, 2023. doi:10.1002/tesj.716.

[5]. C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking Flight with Copilot," *Communications of the ACM*, vol. 66, no. 7, pp. xx–xx, Jun. 2023, doi:10.1145/3589996.

[6]. F. Miao and W. Holmes, *Guidance for Generative AI in Education and Research*, UNESCO, Paris, France, 2023. Available: https://discovery.ucl.ac.uk/id/eprint/10176438/.

[7]. N. Roy, O. Horielko, and O. Omojokun, "Integrating AI Tools in Advanced Computer Science Curricula: A Case Study Of course Redesign," *Proceedings of the ACM Global Computing Education Conference (CompEd 2025)*, Gaborone, Botswana, Oct. 21–25, 2025, pp. 1–7, doi:10.1145/3736181.3747130.

[8]. zyBooks, https://www.zybooks.com.

[9]. Hasan Baig and Phillip Bradford, "Paper-based coding exams: Overcoming coding assessments and grading challenges in the era of AI Large Language Models like ChatGPT", American Society of Engineering Education (ASEE), March 22, 2025.

[10]. Microsoft Teams, https://www.microsoft.com/en-us/microsoft-teams/group-chat-software

[11]. iota, https://www.iota-1.com

[12]. National Association of Colleges and Employers (NACE), https://www.naceweb.org.