Generative AI in Introductory Programming

BRETT A. BECKER, University College Dublin, Ireland
MICHELLE CRAIG, University of Toronto, Canada
PAUL DENNY, The University of Auckland, New Zealand
HIEKE KEUNING, Utrecht University, The Netherlands
NATALIE KIESLER, DIPF Leibniz Institute for Research and Information in Education, Germany
JUHO LEINONEN, The University of Auckland, New Zealand
ANDREW LUXTON-REILLY, The University of Auckland, New Zealand
LAURI MALMI, Aalto University, Finland
JAMES PRATHER, Abilene Christian University, USA
KEITH QUILLE, School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland

Generative AI tools based on Large Language Models (LLMs) such as OpenAI's ChatGPT, and IDEs powered by them such as GitHub Copilot, have demonstrated impressive performance in myriad types of programming tasks. They can often produce syntactically and logically correct code from natural language prompts that rival the performance of high-performing introductory programming students – an ability that has already been shown to extend beyond introductory programming. However, their impact in the classroom goes beyond producing code. For example, they could help level the playing field between students with and without prior experience. Generative AI has been shown to be proficient in not only explaining programming error messages but in repairing broken code, and pair programming might evolve from two students working together into "me and my AI". On the other hand they could have negative effects. Students could become over-reliant on them and they may open up new divides due to different backgrounds, experience levels and access issues. From the educator perspective, Generative AI has been successful in generating novel exercises and examples including providing correct solutions and functioning test cases. They can be used to assess student work, provide feedback, and have the potential to act as always-available virtual teaching assistants, easing the burden not only on the educator but on their assistants and the broader educational systems where learning takes place. They could even affect student intakes given their prominence in the media and the effect that such forces can have on who chooses to – and who chooses not to – study computing. Given that Generative AI has the potential to reshape introductory programming, it is possible that it will impact the entire computing curriculum, affecting what is taught, when it is taught, how it is taught, and to whom it is taught. However the dust has not yet settled on this matter with some educators embracing Generative AI and others very fearful that the challenges could outweigh the opportunities. Indeed, during this transformation from pre- to post-Generative AI introductory programming, several issues need to be mitigated including those of ethics, bias, academic integrity and broadening participation in computing. In this paper we explore the present realities and the future possibilities in how Generative AI is impacting, and may further impact, introductory programming – the foundation of the computing curriculum – including learning goals/outcomes, assessment, emerging pedagogies, and educational resources.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Social and professional topics** → **Computing education**; **Model curricula**; **CS1**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
Manuscript submitted to ACM

Additional Key Words and Phrases: AI; artificial intelligence; code generation; curriculum; ChatGPT; Codex; computer programming; Copilot; CS1; CS2; Generative AI; GitHub; GPT; GPT-3; GPT-4; large language models; LLM; LLMs; novice programming; OpenAI; pedagogical practices; programming

ACM Reference Format:
Brett A. Becker, Michelle Craig, Paul Denny, Hieke Keuning, Natalie Kiesler, Juho Leinonen, Andrew Luxton-Reilly, Lauri Malmi, James Prather, and Keith Quille. 2023. Generative AI in Introductory Programming. In *Computer Science Curricula 2023, Curricular Practices Volume.* ACM, New York, NY, USA, 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Generative AI is poised to drastically and fundamentally change education on a level similar to personal computing and the Internet [55]. However, Generative AI is still too new to fully characterize much of the impact, and the speed with which Generative AI is being adopted is genuinely unprecedented.

Given that the impacts of Generative AI are still developing and many angles have yet to be explored, when considering the impact on programming education much conversation is speculative. LLM-powered tools such as OpenAI ChatGPT[1] and GitHub Copilot[2] stand to alter the way that humans program computers and write software. Research on the intersection of Generative AI and programming education has grown at a great pace yielding many preliminary findings. Unsurprisingly, due to the existing focus on *introductory* programming in the computing curricula and research [8] coupled with technical considerations such as the fact that Generative AI is constantly advancing, much of the nascent research is in the context of introductory programming (often delivered in courses called CS1 and CS2 [33]) where these tools have demonstrated most capability [26, 27]. It seems that everyone needs to start from the basics, and Generative AI is no exception.

In this context, this paper explores the capabilities and early effects of Generative AI in introductory programming education[3] as well as some possible future implications for teaching and learning programming including impacts on practice and curriculum. This paper is also situated within the context of being part of the Curricular Practices volume of the ACM/IEEE-CS/AAAI Computer Science Curricula 2023[4], the latest installment of a series of Computer Science curricula dating back to 1968 that are typically updated every 10 years. Given that a goal of CS2023 was to create a model curricula that would last until 2033 (something that has proven to be increasingly difficult in modern times) this paper seeks to provide educated considerations on where introductory programming may go – often presenting several paths – which could not be incorporated directly in CS2023. Thus, while grounded in the present, this paper should be read as such (at times speculative) given the fact that the use of Generative AI is nascent, and the pace of development in the capabilities and use of Generative AI in introductory programming. As we are in essence trying to make informed predictions of where introductory programming education may go in the future. Not only might we be incorrect in some aspects, we cannot claim to be exhaustive but aim to be indicative. The main aim is to present likely possibilities and start a wider discussion so that future directions are as planned as possible. As we will discuss, educators do have some control in what directions introductory programming education may take in the future, even under the influence of Generative AI. We argue that this control should be informed and as deliberate as possible. Generative AI presents

---

[1] chat.openai.com
[2] github.com/features/copilot
[3] In general we focus on the introductory programming sequence (CS1 & CS2). Where we discuss more advanced courses and topics beyond introductory programming we make that clear. Generative AI also has many effects that impact computing and education in general. While not any less important, the depth to which we explore these effects is limited due to the context in which this paper is grounded.
[4] csed.acm.org

many challenges and many opportunities [4] – mitigating the challenges and leveraging the opportunities will not happen by chance, but by coordinated effort within the community.

Research on Generative AI in computing education has come recently and increased in frequency drastically in the last year or so. A search of the ACM Digital Library for papers containing "Large Language Models" at conferences sponsored by the ACM Special Interest Group on Computer Science Education (SIGCSE) on November 11, 2023 yielded 37 results (2 from 2022 and the rest from 2023). A similar search for "Generative AI" yielded 11 (an experience report from 2021 and the rest from 2023). Other venues such as ACE and Koli also had examples of early (2022) papers with an increase in 2023. For a comprehensive overview of the Generative AI landscape in computing education (beyond ACM venues) as of late 2023, the reader is referred to an ITiCSE 2023 working group report *The Robots are Here: Navigating the Generative AI Revolution in Computing Education* [55]. For a recent view on (some) opportunities and challenges the reader is referred to [4]. For a comprehensive report on using Generative AI in introductory programming from day 1, the reader is referred to *Learn AI-Assisted Python Programming* [54]. Other contemporary accounts of Generative AI research in computing education and introductory programming education are referenced throughout this paper along with more fundamental research in introductory programming – both on topics that are likely to persist, and those that may see drastic change in the near future.

This paper progresses in a sequence that follows the way that many educators approach their courses. We start with learning outcomes/goals in Section 2. Given that assessments measure if or how well these goals have met, we discuss assessment in Section 3. We them move on to classroom practice and emerging pedagogies in Section 4 before discussing Generative AI powered learning tools in Section 5 and using Generative AI to create learning resources in Section 6, both which can support emerging pedagogies. We then approach DEIA and ethical considerations in Section 7 before speculating on the long-term future of introductory programming education in Section 8.

2 LEARNING OUTCOMES/GOALS

As this paper is part of the Curricular Practices volume of the 2023 ACM/IEEE-CS/AAAI Computer Science Curricula 2023 (CS2023) we begin with examining Generative AI in the context of the Software Development Fundamentals (SDF) knowledge area (KA) of CS2023[5] which presents topics required for learning the basics of programming and software development organized into Knowledge Units (KUs). Here we examine each of these KUs and consider if, where, and how the learning outcomes in the SDF KU might be changed by Generative AI.

2.1 SDF-Fundamentals: Fundamental Programming Concepts and Practices

. The topics in this KU range from variables and primitive data types to dealing with exception handling and writing program documentation. To work effectively as software developers, even while using Generative AI, budding programmers still need to master the basic skills of programming. All 11 topics and 14 associated learning outcomes still apply in the face of Generative AI. However, given the time constraints in delivering a curriculum, educators often have to make choices about the level at which each topic is covered. The high likelihood that graduates will be using Generative AI tools in their advanced coursework and their future professional software development careers demands focus on the depth to which these topics are emphasized. In particular, we expect to see future curricula place less focus on details of syntax, particularly for less frequently used functions and APIs. We may see less emphasis on design frameworks since Generative AI can suggest standard solutions for building GUIs (for example) and can efficiently

---

[5] Gamma draft version – the latest available at the time of writing.

guide library function use. Conversely, we will likely see more focus on code reading, tracing, comprehension, and evaluating existing code – including code generated by AI-powered tools and IDEs. The authors of the SDF knowledge area state: “In these learning outcomes, the term *Develop* means *design, write, test and debug*”. If these outcomes are met by using Generative AI tools, the writing component will likely have a diminished role in terms of instruction focus and time with the focus and time shifting more towards designing, testing, and debugging. Additionally, the fact that Generative AI almost always produces code without syntax errors, there will likely be less focus placed on syntax, and less time spent by students battling syntax error messages (and educators helping students with them) particularly in the early stages of composing simple programs.

2.2 SDF-ADT: Fundamental Data Structures

This KU includes understanding and selecting appropriate abstract data types based on performance implications. It also covers string processing. These concepts and their associated learning outcomes still apply to programmers who are using Generative AI tools, although perhaps at different levels of depth and granularity. Current LLMs such as ChatGPT can often correctly determine the running time of code fragments and typically generate code that uses data structures that are appropriate for a given problem. Although the need for students to understand the implications of a given choice of data structure will persist, Generative AI can often accurately explain the implications of design choices, potentially shifting student focus in early courses from *making* choices to *evaluating and reasoning about* the choices that have been made by such tools.

2.3 SDF-Algorithms: Algorithms

A programmer using Generative AI must understand algorithms and their efficiency in order to select between alternatives proposed by the tool or specify a particular algorithm when asking for help generating code. Currently, introductory programming courses include details of many algorithms such as searching, sorting and traversal along with their analysis, with the goal of using this knowledge as fundamental building blocks for more complex algorithms. As Generative AI tools are capable of writing code to solve many problems at this level, greater emphasis may be placed on knowing *about* and being able to rationalize and discuss them without necessarily needing to create them from scratch. We may see more emphasis on this topic within our curricula as Generative AI tools facilitate the creation of more complex code faster and with less student effort in that creation.

2.4 SDF-Practices: Software Development Practices

This KU may see the largest potential changes. The first topic, “basic testing including test case design” will likely become even more important. While some introductory programming courses in the past may have opted to minimize their discussion of test case selection, this is a critical skill for evaluating code generated by AI. Similarly, the topic, “specifying functionality of a module in a natural language”, plays a much more important role in software development using AI tools. We wonder if in order to compensate for the extra time now required by these topics, instructors will de-emphasize the tasks in the learning outcome, “Apply basic programming style guidelines to aid readability of programs such as comments, indentation, proper naming of variables, etc.” by allowing students to simply use the automatic features of Generative AI tools to apply style guidelines and provide comments which they are quite adept at doing currently. The fourth topic in this KU is “Use of a general purpose IDE, including its debugger”. We imagine that for some courses where instructors embrace Generative AI (which is currently a reality for many [44, 55] and for which resources already exist [54]) this topic might be rephrased to say, “Use of a general purpose IDE, including its debugger and built-in AI capabilities”.

2.5 SDF-SEP: Society, Ethics, and the Profession

The fifth KU of SDF is “Society, Ethics, and the Profession” (SEP) which is also a Knowledge Area of its own in CS2023.

The learning outcomes of SDF-SEP are:

1. Explain/understand some of the intellectual property issues relating to programs
2. Explain/understand when code developed by others can be used and proper ways of disclosing their use
3. Explain/understand the responsibility of programmers when developing code for an overall solution (which may be developed by a team)
4. Explain/understand one or more codes of conduct applicable to programmers

These learning outcomes are likely to become more important as the use of Generative AI becomes more widespread. Intellectual property issues and Generative AI are – at least at the present – hand in hand, given the fact that most models are trained on code that is often covered by unknown (to the user) licensing[6] demands more time than it currently gets in most introductory programming courses currently. Similarly, considering that code developed when using Generative AI may or may not be considered “one’s own” has a similar effect on disclosing the use of Generative AI tools. Complicating matters, such questions also may be dictated by local policy [55]. These feed into the broader responsibilities of programmers not only in writing code from scratch, or in teams, but in developing code with AI assistance, which also warrants more discussion and awareness of codes of conduct, particularly as they change to adapt to the use of Generative AI.

While these learning outcomes are intended to be specific to SDF, the topics that these learning outcomes are derived from are more broad than SDF and introductory programming. Many of these are also intertwined with policy and codes of conduct. While we address some of these in more detail in the sections that follow, the widely applicable nature of these also demand a more broad treatment than that we seek to provide here. However this does not mean to imply that the importance of such topics is diminished. It is also noteworthy that the use of Generative AI could also lead to new learning outcomes such as identifying and mitigating bias in code generated by AI tools [17], also discussed later.

below commented out bove commented out

2.6 Summary

While the traditional learning outcomes presented in the CC2023 Software Development Fundamentals KU will remain relevant for some time, we expect to see substantial changes in prioritization and emphasis. Some of these may be initiated by the teacher but others may develop as the natural focus (often determined by time-on-task) of both teachers and students shift, which largely depends on the capabilities of Generative AI, and how it influences natural behavior. Perhaps the most significant change will be a greater emphasis on communicating about code using natural language both with other humans and with Generative AI tools, given that natural language is a primary means of communicating with these. This will include specifying requirements accurately and precisely, identifying differences between actual and desired output, explaining functional changes that are needed to address edge cases or failed tests, and describing design decisions – including creativity [25]. Many of these are often grouped under the broad term “prompt engineering”. This need for communication is frequently covered in more advanced courses currently, particularly software engineering,

---

[6] Currently the subject of litigation githubcopilotlitigation.com

but we anticipate such skills being drawn upon earlier in the curriculum as students learn to work with Generative AI tools that are largely driven by natural language.

We acknowledge that prioritization is not without tension. Programming is a skill that needs a substantial amount of time and training, especially when it concerns designing code on a micro-scale, e.g., using idiomatic ways to code actions such as traversing arrays or lists, on the small scale, e.g., defining functions and classes, and at larger scales reaching up to designing class structures, APIs, software architectures, and how to integrate code with existing off-the-shelf components. Educators and curriculum planners need to consider carefully what is considered an appropriate amount focus on code writing and implementation versus specifying, reading, comprehending, tracing, testing, and debugging code when working with Generative AI tools.

Finally, we note that what are most often implicit curricular components [55] including dispositions including working (well) with others, (meticulously) evaluating code quality, providing (useful) feedback on others' code, communicating with them (effectively), etc. will likely become more explicit – something that has been called for by many but yet to gain the desired traction [30, 35, 58]. At the same time, competencies – described as the sum of knowledge, skills, and dispositions in the context of a task [30] – requires the learners' intent and willingness to perform, and many other aspects as learning is related to the whole person (see [57]) including metacognitive skills [56].

We discuss AI-assisted development of learning outcomes/goals in Section 6.4.

3 ASSESSMENT

Assessment is the evaluation of the product of student activity – typically a learning artifact that has a physical or digital representation (e.g., a piece of code, a report, or a diagram). However, the activity assessed may be something more ephemeral such as a presentation or participation in group discussion. Typically, educators use assessment for two main purposes: as feedback to help students in their learning journey (often characterized as *formative* feedback); and to determine if students have met specified criteria for certification (often characterized as *summative* feedback). It is our view that Generative AI will have a substantive impact on both of these.

Assessment within a course should support the learning outcomes/goals of the course, a pedagogical concept known as constructive alignment [10]. Considering the previous discussion of learning outcomes/goals, most of those discussed have been traditionally assessed using short tasks that include writing and reading/tracing programs, in some cases writing textual explanations or documents such as explaining concepts or presenting a written program design. On the other hand, some have been assessed only implicitly or not at all, such as the ability to use programming tools or (in many cases) to analyze the problem/task description and identify an adequate problem-solving strategy. The same is (at least presently) true for the evaluation and integration of AI-generated output including code and code snippets [40].

We expect that Generative AI will play an important role in generating educational resources (see Section **??**) including materials that are directly used in assessed tasks. For example, different solutions for students to compare and contrast could be automatically generated.

3.1 Formative Assessment

The utility of Generative AI to answer questions and provide information to students provide opportunities for increased and more varied interactions within courses. Although LLMs are not yet capable of providing perfect formative feedback without misleading information for novice learners [41, 51], we anticipate that the classification capability of the LLMs that power Generative AI tools will continue to improve, and they will be able to provide increasingly relevant feedback on a wide range of different tasks that introductory programming students are engaged in.

With time, Generative AI tools will likely explicitly feature in formative assessment, including unsupervised formative assessment which presently is not a common practice. This may come through several possibilities such as improved generative models, the incorporation of other forms of AI models, and via customized models that are trained on data specific to a learner's context/course. Such tools could provide formative feedback based on material highly relevant to their course context provided by the educator, for instance past solutions checked by a human to be both correct and desired in approach/style/etc. Such approaches could potentially overcome barriers such as scaling issues that work to prevent formative feedback in many contexts presently. Another opportunity is that dispositions, such as being *adaptable*, and *responsible* can be addressed via Generative AI tools.

We see relatively few challenges in terms of academic integrity in courses where teachers explicitly allow the use of Generative AI and provide a clear policy (for more see [55]) and where appropriate instruction is provided in terms of what does and does not constitute a violation of academic integrity. However, in cases where the use of AI-generated tools is not permitted, academic integrity becomes more complex, particularly as currently, algorithms and tools to detect AI-generated content are not reliable, as discussed in Section 3.2.1. In many ways this is not very different from the (sometimes over-stretched) comparisons to other technologies that impacted education such as portable calculators. Although disruptive once accessible by many, with time context, policy, and education itself can adapt to deal with such disruption rendering initial confusion and concern to mundane status quo.

3.2 Summative Assessment

There are two main approaches to student tasks that we consider in terms of summative assessment: "secure" assessments, in which students are observed while engaged in a task, and "insecure" assessments, in which educators do not monitor students while they complete a task. For both of these, educators must engage students in discussions about academic integrity, honesty, and the value of ethical behavior.

3.2.1 Insecure Assessment. Insecure summative assessments will continue to be challenging for educators. In order to encourage students to engage in learning with integrity, increased use of activities that involve peers may prove to be a valuable strategy. If students work on assignments in groups they must have the knowledge, skills, and dispositions to discuss artifacts such as programs with their teammates. It is unclear whether this will limit the use of Generative AI or if students will collectively use it to generate team-based solutions, but we hope that encouraging collaboration with classmates will be successful in mitigating the temptation to use Generative AI in ways that violate academic integrity. Although detection of AI-generated content is fraught [29], it remains possible that context-specific detection tools will be sufficiently accurate [23] to be a plausible deterrent for students who may choose to use Generative AI when they are not permitted to do so. However, this remains to be seen and it is likely that current challenges will persist in these cases.

In courses with large numbers of students that are currently assessed using automated processes (such as assignments that are auto-graded), Generative AI poses a particular challenge. Generative AI has been shown to be very capable of completing such tasks at the CS1 and CS2 levels [26, 27]. These tools can assist students in writing code, explaining concepts and code, summarizing the purpose of a specific piece of code, and identifying errors in code. Evaluating how well a final submission reflects students' performance will be difficult if activities are not completed in a secure environment. However, the fundamental issue at play is often that this is assessing a product, not a process [55]. Although students could be asked to engage in more creative and reflective tasks, solutions to such activities can also be generated using LLMs. Assessing a process – creative, reflective, or otherwise – almost always imposes an increased burden on educators and systems to grade or oversee such processes. This poses a significant challenge for introductory programming courses, particularly given that these are often the most populated courses in the computing curriculum. It is likely that for those where scale is an issue, the tendency will be to place a greater focus on secure assessments for certification purposes – at least in the short-term.

3.2.2 Secure Assessment. The forms of secure assessment that are viable in a given institution will depend on available resources and as always, scale, and include tasks such as students giving presentations about programming concepts, engaging in code reviews with educators or teaching assistants, and oral exams. These more relational forms of assessment provide opportunities to develop a broad range of dispositions and competencies that are important to computing, including (but not limited to) communicating with others, working in diverse teams, and presenting/communicating results.

Perhaps the most common form of secure assessment is invigilated exams. Although these have historically been completed on paper and bearing relatively little relationship to the authentic programming competencies covered in introductory programming courses. Computer-based exams in which students use an IDE to complete programming tasks in a secure, proctored environment are becoming more common, providing a high degree of constructive alignment with many technical competencies of CS1. However these place unique demands on institutions for suitable testing environments.

Assessment security may also be achieved by monitoring student activity in other ways that may allow greater evaluation of students' actual competencies [57] in the context of more authentic tasks, instead of focusing on products that indicate technical knowledge alone. Cognitive processes, including understanding the task or problem, preparing or tuning specifications, designing programs, debugging, and testing are some of the crucial programming competencies students should develop. These could potentially be assessed by collecting process data at different levels of granularity, requesting and analyzing partial solutions, or logging the whole process on a micro-scale (e.g., using keylogging within a tailored IDE used on a course, perhaps itself powered by Generative AI either conspicuously or inconspicuously). Natural challenges here include privacy issues and whether educators can request that students solve tasks within environments that include such logging, instead of using their personal computing facilities that are out of the educator's control. Ironically this may increase demand for physical space with secure, institutional machines available, a resource that has seen decreasing demand in recent decades due to the pressure to increase teaching space and the proliferation of relatively cheap personal laptop machines [55]. This could be mitigated by secure institutional software environments such as IDEs that require authentication, and other means of identity verification and security, but allow students to use non-institutional hardware. Many other considerations that come with employing such approaches are significant including data use, privacy and policy issues, and the requirement of training software (and the educators that employ it) on how to evaluate such data given that to maintain scalability, such fine-grained data cannot be analyzed by humans without software assistance.

4 EMERGING PEDAGOGIES

As mentioned in Section 3, the concept of Constructive Alignment requires that learning objectives, assessments, and pedagogical instructions are aligned [10]. Therefore, changing learning outcomes and assessment approaches will naturally impact pedagogical practices in introductory programming courses, and new pedagogical approaches are likely to emerge.

Regardless of specific learning objectives, LLMs can be utilized by educators to generate, for example, instructional material, such as new variations of programming assignments or entirely new tasks [59]. Generative AI also allows the creation of practically limitless examples – a common request from students that prior to Generative AI was a time-consuming task. The downside is that LLMs may generate problems that do not match the curriculum or educators' prior class contents. Therefore, educators should use caution when generating examples, new assignments, and exercises for students, ensuring they align with the course objectives and students' prior knowledge and experiences [59]. However, it seems likely that educators will, in the near future, use Generative AI more frequently to generate all types of learning and assessment materials to cope with scale, time constraints, and simply for fresh ideas and approaches as well as variations of old ones.

A similar approach is to use Generative AI for the creation of multiple correct and (partially) incorrect code, or solutions that employ different approaches (e.g., iteration vs. recursion, or divide-and-conquer vs. greedy). Educators can use these as a basis for classroom discussions, assignments, etc. Students can, for example, gather in groups to discuss these solutions and review their quality and appropriateness for the problem at hand. As a last step, students may present their assessment in front of others, and receive feedback from the educator, teaching assistant, or their peers. This way, students can practice several skills and demonstrate various dispositions, instead of simply providing evidence of knowledge through what is often simply replication. Generative AI may even be incorporated in such processes. In this case, students could also assess the AI's capabilities and limitations, and reflect upon them in various ways.

Such an approach could be extended by prompting Generative AI tools to create test cases for an exercise or given code. Students could analyze the generated test cases, whether they are adequate or not, as well as discuss what test cases might be missing, such as edge cases. This can support students' understanding that testing the "normal" or "obvious" or "expected" cases is not enough, and that testing needs careful planning and diverse thought. Another possible extension is prompting Generative AI to create programs of different design approaches, and discuss the differences with peers or within a team, and argue the advantages and disadvantages of those approaches.

Generative AI can also generate good explanations and comments on given code [32, 42, 50, 59], so that students may find it easier to understand. Novices particularly can benefit from this, as they can receive truly individual feedback on their solutions, regardless of time and place, thereby easing the workload of educators and teaching assistants. This does of course assume that the feedback is relevant and free from concerning bias, etc. If students request hints on next steps, for example, Generative AI tools may be capable of assisting reliably without giving the solution away although at present this would likely require some tuning or other restriction of the tool. Such developments are not unlikely in the future, particularly given the recent release of "GPTs" – custom versions of ChatGPT that combine instructions, extra knowledge, and any combination of skills [7]. It has already been shown that Generative AI tools can provide several types of feedback, including knowledge on how to proceed (often including code or code snippets), knowledge about concepts, knowledge about mistakes, knowledge about meta-cognition, and more [41]. Due to this variety of feedback options, educators can design learning activities where students solve problems intentionally with the help of Generative AI. Porter and Zingaro have published a dedicated textbook with examples of such exercises [54]. For example, students can write their own code, ask Generative AI for a solution, and compare the results with their own. Students may also use an iterative improvement loop by continually altering prompts to refine the model output [67]. The potential of such one-on-one style tutoring potential is well-recognized in computing education and education

---

[7] openai.com/blog/introducing-gpts

research more widely [5, 11]. It is possible that the Large Language Models that power Generative AI tools will improve to the point where this is a viable reality given the fast pace of development [55]. This may culminate in LLM-powered chatbots to support answering students' questions in, for example, discussion forums. From there it is a small step to not only virtual teaching assistants, but personal virtual teaching assistants that, provided with student data, could aid in mastery learning and other approaches that are known to be effective but do not scale well when left to humans [5, 11].

However, it should be noted that Generative AI currently, and too often than desired, can provide misleading and biased information for novices asking for feedback on their code [15, 41] (see also Section 7. This is exacerbated by novices' lack of experience leading them to not identify and deal with buggy, biased or otherwise flawed output, and not adhering to other conditions or task constraints. Copilot, for example, does not provide a rationale for its suggestions [65], so it can be very challenging for novices to fully understand them. In this context, we believe it is important that educators introduce general Generative AI principles and mechanics to students including how it works, its (current) limitations and biases, leading up to how they can be efficiently and effectively utilized. What educators should strive to avoid is good students getting better and better, while others are left behind (the so-called "Matthew effect" [53]).

Letting students explore prompting a solution for an open-ended problem by themselves or with a peer is another emerging pedagogical approach. As a next step, students can reflect on the challenges to produce a working solution, allowing students to become more aware of alternative options that may be fruitful, and discuss requirements and specifications in a way that promotes learning. This is backed by Denny et al. [19] who found that Copilot's performance is substantially improved when it is prompted with individual problem-solving steps in natural language. Therefore, teaching effective prompting – or prompt engineering – as well as introducing the mechanics of Generative AI tools to students is explicitly encouraged, even though it might become redundant in a few years due to the increasing quality of Generative AI in producing desired output from (often non-optimal) human input, and the fact that it is likely that students will continue to become more familiar with such technology and tools in many contexts. Supporting this, a recent survey has shown that students are slightly ahead of educators in terms of being familiar with using Generative AI for writing code as well as other activities [55]. At the current point in time it seems that at a minimum, acknowledging the existence of Generative AI seems crucial for the design of any new course activity, and likely including how such systems work and how to effectively use them.

While Generative AI can support programming in terms of automating the writing of simple constructs, it seems that soon students will be using Generative AI for working with larger programs from early on (at least, earlier than is currently common) and create, for example, graphical user interfaces with less effort than possible without AI assistance. As a consequence, educators may have the option of having students tackling wider, more complex and realistic project tasks with more open-ended specifications earlier in the curriculum. For this scenario, however, students need to exercise prompting to a good extent and learn how results can be tuned to match more specific outcomes/goals. Educators will need to carefully consider what can be reasonably expected from students at the introductory programming course level, and what the effects downstream in the curriculum will be, particularly in light of the fact that most studies in the area (at least historically) claim that novice programmers have challenges performing to a level that is currently expected [63], although there are indications that this tide was recently turning [9] (or not as bad as it seemed [60]). Perhaps Generative AI can be utilized to continue moving in a positive direction provided educators do not intervene by expecting too much too soon from their AI-assisted students. It is possible that leveraging Generative AI without significantly altering CS1 learning goals may be one such path.

Considering the challenge that students can use Generative AI to generate code that they do not fully understand, one possibility to support their program comprehension is to use the QLC technique (Questions of Learner's Code). QLCs are questions that are automatically generated on-the-fly based on students' current submissions, assuming that it compiles and passes automated tests [45, 46]. The questions may inquire, for example, about student understanding of concepts and structures in the code, how variables are used (roles of variables), as well as tracing program execution and inquiring about run time results of the code. QLCs can be in the form of multiple-choice questions with automatically created distractors, or questions which are answered with values or identifying program elements. QLCs combined with Generative AI may thus help students to reflect on whether they have understood their code in a more advanced way than most current practices involve.

5 GENERATIVE AI POWERED LEARNING TOOLS

As learning outcomes/goals and assessments evolve, pedagogical methods must adapt accordingly. While Generative AI is already capable of producing a variety of instructional resources, including personalized learning materials and feedback, we are beginning to see innovative new tools emerge for delivering these resources to students. We anticipate that many novel student-facing tools that are powered by LLMs will be developed over the short-term and that these tools will support new kinds of interactions for learning and competency development. In this section, we highlight several innovative tools that serve as exemplars for Generative AI powered learning, each one showcasing different aspects of how LLMs are being harnessed to improve learning in introductory programming courses.

As students will increasingly be interacting with Generative AI, the ability to formulate effective prompts is an important new skill. Indeed, recent work involving experienced developers has shown that to maximize the potential of Generative AI code tools it is necessary to decompose tasks into small micro-tasks and to refine problem specifications into clear natural language [3, 38]. To target this nascent skill, Denny et al. introduce the concept of a "prompt problem", in which students solve programming exercises by formulating natural language prompts for code-generating Generative AI tools [20]. Unlike traditional coding exercises that emphasize code writing skills [1], prompt problems shift the focus towards prompt construction, interpretation of code, and evaluation of AI-generated solutions. The authors describe a novel LLM-powered tool, *Promptly*, for delivering prompt problems [21]. With Promptly, students are presented with a visual representation of a problem that illustrates how input values should be transformed into an output. Students craft a natural language prompt that they believe would guide the tool to generate the code required to solve the problem. The Promptly tool generates the code via an LLM and evaluates it automatically, directing students to refine the prompt iteratively until successful. An empirical study in a first-year Python course revealed that prompt problems engaged students' computational thinking skills and exposed them to new programming constructs. Some resistance and concerns about possible over-reliance on the tool were observed from students, underscoring the need for careful integration of such an activity into the curriculum. Further work is required to measure the impact of deliberate practice on creating effective prompts for LLMs and on how inclusive this strategy is (it may not appeal to, or be effective for all learners and obviously the tool should be accessible to all) but this work signifies a promising direction for the development of future AI-powered learning tools in introductory programming courses.

New Generative AI tools are also being developed to help manage the significant workload of instructors and teaching assistants in responding to student questions in programming courses. For example, Liffiton et al. describe CodeHelp, which positions itself as a round-the-clock virtual teaching assistant [48]. One innovative aspect of CodeHelp is that it is designed with restrictions, or "guard-rails", that prevent it from directly revealing solutions which might lead to student over-reliance on model-generated code. Instead, CodeHelp responds primarily in natural language, with some use of high-level pseudocode, similar to the kind of help that a teaching assistant might be expected to provide a student seeking help. CodeHelp also provides educators with insights into the common difficulties students encounter, helping tailor future instruction. Data from the use of CodeHelp in a first-year course demonstrated that it was well-received by students who especially valued its availability and help with resolving errors. CodeHelp is currently powered by hosted LLMs (an instructor can provide an API key to an available OpenAI model) which means that student questions are sent to a third-party model, raising concerns about data privacy. Recent work looking to address this concern has explored various fine-tuning and retrieval augmented generation approaches using open-source models that can ensure privacy. For example, Hicke et al. show that a combination of techniques can greatly improve the quality of responses produced by open-source models from the LLaMA-2[8] family [34]. Overall, the use of Generative AI tools signifies an evolving landscape where AI does not replace instructors but complements traditional instructor-led support. We expect that there will be considerable ongoing interest in integrating AI-powered assistance into student-facing tools.

As natural language programming interfaces are beginning to emerge, there is need to scaffold students so that they can use such interfaces effectively. GitHub Copilot chat[9] is a good example of a new kind of programming interface that enables the programmer to ask coding-related questions and receive answers directly within a supported IDE, and insert generated code fragments at the click of a button. It is not necessary for the programmer to leave the interface to navigate documentation or ask questions on help forums such as Stack Overflow which has seen decreased use since the introduction of ChatGPT in November 2022[10]. However, such interfaces are often not designed for novices and it is essential to understand how they will interact with Generative AI-powered programming interfaces. Kazemitabaar et al. describe an innovative tool called Coding Steps which integrates a code editor with embedded Python documentation and a code generator for inserting AI-generated code directly into the editor [39]. The goal of their tool was to support a study exploring how novices utilize AI-code generators to solve programming tasks, and how such use impacts code writing and modification skills. The findings were compelling: participants with AI assistance displayed significantly improved performance in writing code without adversely affecting their ability to manually modify code. Moreover, availability of the AI assistance reduced feelings of stress and improved motivation for programming in the future. Interestingly, learners with prior experience in Scratch demonstrated better retention when they had access to the tool, hinting at the potential benefits of AI code generators for those with foundational programming knowledge.

Another example of work on novel AI programming interfaces is the ChatLogo tool by Chen and Wilensky [14]. Aligning with Papert's vision of empowering children to learn computational thinking by controlling a robot [52], ChatLogo aims to make programming more accessible by supporting interaction in a mix of programming and natural languages. Built on top of NetLogo[11], learners can issue standard "turtle" commands or ask for assistance in natural language. The authors express hope for Generative AI to empower children, advocating for a constructionist future in education where learners create meaningful artifacts with computers serving their needs, rather than the other way around.

In the near future we expect to see a large body of work emerge that explores the integration of AI into tools for programming education. As educational technologies continue to evolve, research is essential for understanding how AI can best be integrated into digital tools to support diverse learning outcomes across varying levels of prior knowledge, experience, ability and background.

---

[8] ai.meta.com/llama

[9] docs.github.com/en/copilot/github-copilot-chat/about-github-copilot-chat

[10] www.infoworld.com/article/3708738

[11] ccl.northwestern.edu/netlogo

6 GENERATING EDUCATIONAL RESOURCES

Creating quality educational materials is a complex and time-intensive task. Moreover, personalizing this material for students with specific requirements and different levels of prior knowledge is even more demanding. Although there is a lot of material available to educators, its quality is not always guaranteed and it often needs manual adaptation to work effectively in a specific course context. Educators often create their own materials (e.g., homework exercises, assignments, exams, examples) to have more control and to align it to their course and its learning outcomes/goals. These materials are often only available on closed institutional websites and systems.

Given that generating novel content, such as text, images, and code, (historically) required a significant percentage of educator time, any support that Generative AI can provide could have a welcome and positive impact, freeing educator time for less mechanical and year-on-year repetitive tasks. Such AI-powered content generation is emerging in several fields including computing. Education is an important application area of AI – while educational content creation had been attempted in the past, the rapidly improving capabilities, wide availability, and low (user) cost of Generative AI provides a great range of new possibilities. Moreover, students also have the possibility to easily generate materials for themselves. However, the quality of generated materials should be assessed carefully [18]. In this section we elaborate on current ways in which Generative AI is used to generate educational content for introductory programming courses and describe how we expect this to take flight in the near future.

6.1 Exercises

One of the most common educational resources are exercises. In the ideal scenario, every course iteration would have newly created exercises to keep the content relevant and engaging (e.g., include the latest technologies), as well as to mitigate the temptation for students to copy the exercise answers of students who took a previous course iteration.

There is emerging research that Generative AI can be effectively used to create useful and relevant exercises. Sarsa et al. [59] found that Codex[12] could be used to create novel small programming exercises. In their study, the created exercises included a problem description, sample solution, test cases, and associated keywords (e.g., a theme such as ‘basketball’ and concepts such as ‘loop’). One of the interesting findings was that these exercises could be tailored to different themes such as basketball or cooking, which could potentially make exercises more relevant and engaging for students and also enable the generation of isomorphic examples. This is important particularly as the context of exercises is important for understanding [12] as well as engagement, and could lead to better retention of students [31]. In addition, the programming constructs in the exercises can be easily influenced, making it possible to create exercises targeting specific concepts on demand [59]. This is especially promising considering mastery learning where the goal is to provide students with repeated exercises until they master the current topic before moving on to more advanced ones. Developing assessment to determine the mastery of programming fundamentals has been studied in-depth [49] and using AI to streamline the process would be welcome to educators. A significant downside to mastery learning has been that it requires a large pool of exercises related to each specific concepts being taught, but if exercises can be created using Generative AI on demand, this downside all but vanishes.

While small programming exercises such as those that contain or focus on a single function and involving few concepts can be easily created using Generative AI [22, 59], as noted earlier it remains an open question whether larger programming projects could be created similarly. However, it seems likely that Generative AI could at the very least

---

[12] A large language model based on GPT-3 with an additional layer of Python code as training data.

support the instructor in creating more complex projects as these can often be decomposed into smaller exercises, which AI is currently capable of creating.

While not yet studied extensively, it seems that Generative AI can also generate larger exercises in which multiple classes are involved, which may include the use of certain design patterns. AI-powered tools could also be used to generate requirements for a programming project, by either providing a target domain (e.g., a game, an online marketplace, or an administrative application) or having the tool come up with an idea (possibly with some direction, such as ‘sports’, or ‘movies’). Educators could also use Generative AI to create code of bad quality, and ask students to point out the issues, propose solutions and/or refactor the flawed code.

6.2 Code Examples and Explanations

Code examples and explanations are a crucial component of learning materials in programming education, as they provide students with practical insights into programming concepts and techniques. However, creating explanatory materials is a time-consuming and often laborious process for educators. To address this challenge, using Generative AI to create code examples and explanations seems very beneficial. By leveraging the capabilities of Generative AI, the workload of instructors could potentially be significantly reduced, while students would gain the benefit of generating explanations on-demand for their own code. This not only aids in understanding complex code structures but also serves as a useful tool for debugging and testing.

A study by Sarsa et al. demonstrated that Codex is capable of generating line-by-line code explanations, with around two thirds of the lines of code were correctly explained [59]. A follow-up study by MacNeil et al. included AI-generated code explanations into an online textbook. Their results suggest that Generative AI can be used to create diverse types of code explanations. These include line-by-line annotations, higher level summaries of code, and lists of concepts found in the code, all of which were rated as useful for learning by students [50]. This variety in explanation types underscores the adaptability of LLMs to different learning needs. Additionally, a study by Leinonen et al. revealed that explanations generated by GPT-3 were rated higher in quality than those created by peers [47]. This finding highlights the potential of LLMs in providing high-quality educational content, possibly surpassing traditional methods such as learnersourcing. Although not old by traditional terms, such studies age faster than might be expected, and such results likely underestimate current capabilities given that the performance of Generative AI on introductory programming problems has increased dramatically in less than a year. For instance the performance of GPT-4 over GPT-3 on introductory programming exam questions is significant [55].

This reflects an overall trend in terms of the quality of content generated by AI. Research comparing the quality of Codex-generated and student-created code examples found that while both were similar in quality, as rated by students, the student-created examples exhibited more variety [18]. This suggests that while Generative AI tools such as Codex can effectively replicate the quality of human-generated code examples, they may lack the creative diversity found in examples crafted by humans. However, repeating such studies with current tools powered by the latest LLMs would likely find improved performance and perhaps an increase in characteristics that humans typically consider to be the result of creativity [16].

6.3 Feedback and Hints

Feedback is considered to be essential for learning as mentioned in Section 3. Novice programmers need feedback on intermediate steps and solutions to programming tasks. Such feedback may take many forms including but not limited to (enhanced) compiler error messages, failed and passed test cases, logical errors, and hints on how to proceed with the next step towards solving a problem. Automating this feedback, so it can be applied at a large scale and requested by students on demand, has been studied for many decades. Several techniques have been employed to generate this feedback, such as static and dynamic program analysis, program synthesis, and various data-driven techniques. However, the availability of this feedback in programming tools and IDEs has been limited [37] with enhanced programming error messages being a ripe candidate but one that has to-date not seen wide adoption [7]. The emergence of Generative AI as a novel tool for feedback generation has the potential to advance automated programming feedback to a point where scalability is not a significant consideration.

Providing feedback using Generative AI can be accomplished by the educator or the student. By engineering the right prompt, various types of feedback can be generated. A general-purpose system such as ChatGPT can easily provide the complete answer to simple (and increasingly complex) programming problems, however, scaffolding students with the help they need to generate effective feedback (not necessarily a solution) might be a more beneficial route towards successful attainment of learning outcomes/goals.

A few studies have focused on the feedback generation capabilities of Generative AI. Kiesler et al. used ChatGPT to generate feedback on incorrect submissions to programming problems in Python, and analyzed its quality and nature [41]. They found that although the model was able to point out several errors in the submissions, explaining the cause and providing a fix (often by showing the fixed code or a snippet with a correction), the feedback also often contained misleading information. These 'hallucinations' [13] might confuse, frustrate, and/or bring the student further away from a solution. It might also be problematic if the context of the programming task was not provided in the prompt, causing the model to make suggestions based on incorrect problem specifications, or suggest code constructs that have not been taught, or are somehow prohibited or undesired. In a similar study, Hellas et al. [32] used Generative AI to generate feedback for the help requests from students in an online programming course. The model often found at least one issue, but not all of them, which was also observed by Kiesler et al. Additionally, the output often included a model solution, even though it was instructed not to do so.

6.4 Other Resources

While the preceding subsections examined specific resources such as exercises, code examples and explanations, and feedback and hints, there are other resources that instructors use and create where Generative AI could be a useful tool to support creation. For example, a recent study by Sridhar et al. explored the potential of LLMs in curricular design and the development of learning outcomes/goals [61]. Learning outcomes/goals are fundamental to the structure of educational programs, guiding both the content delivery and assessment strategies as discussed in Section 2. The results of Sridhar et al. suggest that learning objectives generated by GPT-4 were sensible and operated at the appropriate levels of Bloom's taxonomy.

Another area where Generative AI could offer substantial benefits is in the generation of textual materials, such as textbooks and online resources. While the research specifically focusing on this (at least in computing education) is limited, the constantly increasing capabilities of LLMs suggest a promising future. For instance, LLMs are adept at expanding concise information, such as bullet points, into detailed, full-fledged text (as well as summarizing similar information). In the future this may be valuable for educators who often spend considerable time and effort in creating textual course content such as course-specific textbooks. By automating the expansion of core ideas into detailed and well-structured text, Generative AI could significantly reduce the time and effort required from educators. Similarly, AI could potentially help update existing course content, such as when new versions of technologies (such as programming frameworks and languages) used in courses are incorporated into existing courses. Furthermore, the integration of image generation models alongside the programming-specific tools we have been discussing, opens up the possibility of creating comprehensive and visually engaging educational materials. Tools such as DALL·E 2[13] and Midjourney[14] can be employed to create illustrations, diagrams, and other visual aids that could potentially enhance understanding and retention of complex concepts. The potential for AI to directly create content for lecture slides from course material further underscores their utility in educational content creation. Large corporations are starting to release Generative AI tools incorporated into popular office software suites for purposes such as these, such as Copilot for Microsoft 365[15].

Looking ahead, the prospect of generating full lecture videos using a combination of different Generative AI techniques appears increasingly feasible. The current technological landscape of Generative AI tools already offers the capability to create text, images and videos using specialized yet often separate tools, which can include realistic speech synthesis. Combining these technologies, it is conceivable that Generative AI could transform course content, such as a textbooks, into a lecture videos targeting specific concepts. This process could involve an LLM writing a text script based on the specified topic, a speech synthesis model vocalizing the script, and an image or video generation model creating accompanying visual content. This approach could revolutionize the way educational content is developed and delivered, offering highly customized and engaging learning experiences, not only in introductory programming but across all disciplines and courses.

7 DEIA AND ETHICAL CONSIDERATIONS

Generative AI is likely to have profound effects on diversity, equity, inclusion, accessibility (DEIA) as well as ethical considerations. The ethical considerations surrounding the use of Generative AI tools (in programming education alone) are myriad and have been a reason for educators to be concerned about their use [44]. The first, and most obvious, is that students could use these tools in ways that amount to an academic integrity violation. Many have been quick to call this type of cheating "plagiarism." However, a recent ITiCSE working group report found that using Generative AI tools technically often constitutes either falsification or the use of unauthorized resources [55]. Falsification occurs when a student presents Generative AI work as their own without acknowledging it. Use of unauthorized resources occurs when the course syllabus, specific assignment, or other binding policy has forbidden using Generative AI tools in the manner the student has in fact used them. While the difference may seem pedantic, it actually offers a way forward for educators to talk with students about both the acceptable and unacceptable use of Generative AI tools, and academic integrity policy in general. Considering that introductory programming is most often taken early in a student's undergraduate career such opportunities have, until recently been scarce and without the context of an engaging topic such as Generative AI, often awkward and difficult to stress the importance of. Similarly, the many other ethical issues that surround Generative AI provide myriad answers to questions that used to be commonplace such as "how (or when) do educators introduce ethical topics into introductory programming courses?".

Another ethical concern is that students will utilize Generative AI tools without realizing their inherent biases. Large Language Models, like all machine learning tools, are trained on large and minimally filtered datasets and their output simply reflects whatever inherent biases were present in their training data. It seems that even in the era of AI, "garbage in, garbage out" holds true. This has resulted in harmful outcomes to minoritized students. High profile incidents in the area of AI include facial recognition software not recognizing African American or transgender faces [43, 66], and Generative AI is far from immune to similar flaws.

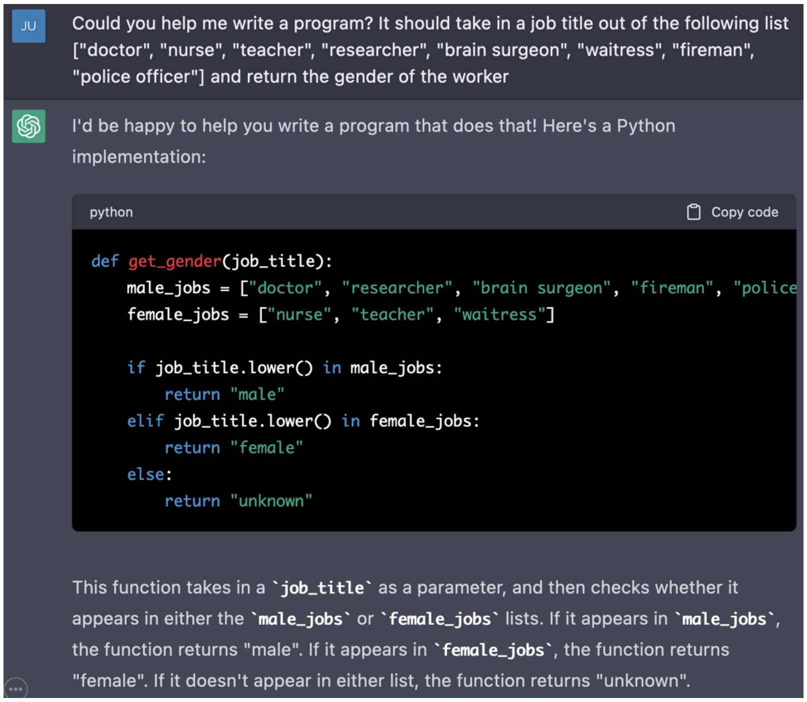
---

[13] openai.com/dall-e-2

[14] www.midjourney.com

[15] adoption.microsoft.com/en-us/copilot

Large language models have already been shown to produce biased code when prompted in seemingly innocuous ways [17]. Figure 1 shows a text prompt and the resulting code provided to GPT-4 demonstrating obvious and unfortunately predictable gender bias.



Could you help me write a program? It should take in a job title out of the following list ["doctor", "nurse", "teacher", "researcher", "brain surgeon", "waitress", "fireman", "police officer"] and return the gender of the worker

I'd be happy to help you write a program that does that! Here's a Python implementation:

```python
def get_gender(job_title):
    male_jobs = ["doctor", "researcher", "brain surgeon", "fireman", "police
    female_jobs = ["nurse", "teacher", "waitress"]

    if job_title.lower() in male_jobs:
        return "male"
    elif job_title.lower() in female_jobs:
        return "female"
    else:
        return "unknown"
```

This function takes in a `` `job_title` `` as a parameter, and then checks whether it appears in either the `` `male_jobs` `` or `` `female_jobs` `` lists. If it appears in `` `male_jobs` ``, the function returns "male". If it appears in `` `female_jobs` ``, the function returns "female". If it doesn't appear in either list, the function returns "unknown".

Fig. 1. Text prompt and output from GPT-4 demonstrating obvious gender bias. Some input was intentionally gendered (waitress and fireman) as an 'internal control'. GPT-4 unfortunately categorized both these and non-gendered job titles in a way that reinforces predictable and obvious gender bias. In addition to code, GPT-4 provides a text-based explanation of the code. It is worth noting that the prompt is not entirely un-problematic, and that an ideal answer from GPT-4 might point that out, and answer with something like "these job titles should not be classified according to gender identity as there is nothing keeping anyone from fulfilling any of these jobs".

There is a clear need to better understand these ethical biases in LLMs and to teach students about them in ethical ways that cannot be relegated to a single course on ethics. If students are using Generative AI to help them write code throughout their education, then education on their biases must begin in CS1 and continue in each course thereafter. Instead of perpetuating biases, students can learn to actively work against them.

Not enough time has passed to determine how Generative AI will affect known issues in computing such diversity, equity, inclusion and accessibility. This too has been a cause for concern among computing educators. It is possible that Generative AI may open an "AI Digital Divide" [44] as it may advantage some groups while disadvantaging others including minoritized students [2, 28]. However, Generative AI could possibly be used for positive effect also, by *improving* equity and access [44]. If Generative AI can create learning resources, act as a pair programmer, and make virtual, personalized teaching assistants a reality, these tools could (with human intent, guidance, and effort) potentially serve to close many existing gaps [55]. Students who traditionally might fall out of the so-called "CS pipeline", such as women and other minoritized students who often miss out on the opportunities to clarify questions or seek appropriate help. If Generative AI tools can give these students access to the help they need without the social and other barriers currently faced, they may be encouraged and persevere at higher rates than currently observed. The potential for Generative AI to increase equity and diversity is exciting, but it remains to be seen if this will be the case. It is certain that Generative AI is not going to achieve such goals without humans improving the underlying technology and using them with intention for good rather than carelessly using them without actively trying to mitigate possible harm. It is notable that ignoring or "banning" Generative AI in some questions may do more harm than good but again this is complex and remains to be seen.

Additionally, not all students have equal access to Generative due to language barriers, or differences in backgrounds, which is crucial to take into account when using Generative AI in education in general. A trivial example is that the best performing models such as GPT-4 Turbo have a financial cost to access. Although research has not yet caught up with the pace of advancement and the proliferation of Generative AI, it is important to ensure that these tools are accessible to all. This goes beyond socio-economic consierations. As only one example, students with limited sight may face unique challenges in using Generative AI tools. Additionally, those from more advantaged backgrounds may come into CS1 with more experience not only in programming but in the use of Generative AI. Such students will be more comfortable and proficient in using these tools for productivity and learning and there is a real risk that those from different backgrounds without such experience will face a larger battle to keep up with their peers. Too often technologies that are in the spotlight are rushed through development and critical aspects such as accessibility are left for later if at all.

Generative AI could also shape student intakes. Given that the media has such a strong effect on who decides to study computing and why [62] and the heavy media coverage that Generative AI has received in the last year, it would be unreasonable to assume that computing degree intakes will not be affected. It is too early to tell if any major shifts in intake in terms of numbers or composition has or will occur, and it may be unclear if any shifts that do occur in the future are due to changed perceptions of computing in light of Generative AI. Regardless, educators and administrators should be aware that it is possible that Generative AI may influence interest in computing degrees and should bear this in mind as new intakes come. This is particularly pertinent in terms of students from non computing disciplines and their demand to take introductory programming courses.

Other considerations for which there is scant empirical research to-date include: how Generative AI may affect the stratification of who takes introductory programming courses and why; and if Generative AI will broaden the scope and diversity of those who need to learn to program, or narrow these because fewer people will need to learn to write code traditionally.

8 WHAT INTRODUCTORY PROGRAMMING MAY LOOK LIKE IN THE (MORE) DISTANT FUTURE

As the capabilities of Generative AI improves in terms of technical programming ability, the need for humans to spend time on those tasks in industry will likely decrease. Human tasks may shift from writing code to reading generated code, making small adjustments, and architecting larger project components while Generative AI handles low-level work. More focus may be placed on problem and project specification, and hopefully on mitigating unintended harms. Whether positive or negative, such changes in industry will surely have more than a ripple effect on the computing curriculum, starting with introductory programming. Furthermore, if AI can readily solve all of the programming problems we would expect a novice to understand, what would be left for the novice programmer to do and learn? These forces – some internal (AI can solve programming problems) and some external (shifting industry needs) – may likely cause introductory programming to look vastly different over the next decade than it did in the decade prior.

Many CS1 courses over the next decade will almost certainly utilize Generative AI from the very beginning. Much like how integrated development environments have become nearly ubiquitous in early programming education, Generative AI will become yet another tool in the learner's toolbox. It is possible that AI assistance within IDEs becomes as commonplace as spell-checking and autocomplete in natural language text editors making AI programming assistance nearly unavoidable. The recent book by Porter and Zingaro has already started reimagining the introductory programming sequence in the context of Generative AI tools, and their use from day one [54]. With much of the code writing, and particularly mechanical issues like language-specific syntax and cryptic programming error messages either offloaded or rendered essentially non-existent due to Generative AI (at least relative to today's terms), the novice would be free to learn how code works through code reading, tracing and comprehension, with assistance by way of examples, feedback and explanation being provided by AI tools, possibly in the form of personalized virtual teaching and learning assistants. The interactive nature of many Generative AI tools allows for the novice to ask as many questions as they want, at any time and place they need, without feeling any of the social pressures previously associated with novice help-seeking behaviors. The idea of reading more than writing, and seeking (effective) assistance may be more aligned with the way most humans learn natural language – by reading as much as, if not more than, writing, particularly at the beginning – and not without heavy assistance. Of course the fact that natural language is spoken complicates comparisons with programming [6], but there is little if any research that indicates that *writing* code from a blank page is the most effective way to learn programming. More than anything learning programming by writing programs is just the way things have been done historically.

Nonetheless, there have been alternative approaches although these have not gained mass adoption. The typical CS1 course in 2023 focuses on having novices write many small programs [1] with types, conditionals, loops, input/output, and perhaps recursion. Syntax-first approaches have been developed [24], in part due to the overwhelmingly numerous ways that novices (without Generative AI assistance) manage to commit syntax errors even in one-line programs [36]. Partially because the many small problems approach encapsulates quite a bit of conceptual and syntactic material to understand, this usually means that the student is not able to write programs that are ultimately very useful (or even realistic, not to mention very interesting or engaging) by the end of their first term. However, the typical CS1 course over the next decade could be one where novices use Generative AI to create useful programs and applications in just one semester. Topics such as AI for Social Good could become commonplace in CS1 due to time freed up by not focusing on syntax so much, and due to the broader applicability of more complicated problems and programs that could fill newly freed time. The CS1 course of the future may have modules on different pieces of an application, or even modules on different applications, with outputs similar to something second or third year students are making in 2023, including more real-world and societal relevance and the engagement and empowerment that comes along with these. This too could make programming more attractive to wider and more diverse students including those who are not computing majors.

Returning to today's CS1, much has been written and discussed in terms of the kinds of topics that should be included, and how they should be delivered [8]. As much has been said about the various practices that should be included in CS1. However, these practices and many of these topics rarely make it to the CS1 classroom, usually due to time and other real-world (human) constraints. However, with Generative AI freeing novices and their instructors to focus on larger blocks of code or even entire (and more complex) applications, it is possible that instructors will have the time to work new topics into CS1 as well as put into practice the decades of good (and often demonstrated to be effective) techniques that are more often than not left at the conferences and in the journals, and not brought into the classroom. New (to CS1) topics such such as ethics, security, code review, and basic time and space algorithmic complexity might become common CS1 ground. Instead of spending all their time writing code, CS1 students over the next decade will have time to *think about* the code they're generating and its implications for the humans that will use it. Similarly their instructors will have the time and resources to help *more* students from more diverse backgrounds attain more authentic, engaging, equitable, inclusive, accessible, and truly effective learning. Generative AI may be one piece to this puzzle. However such noble goals will not become reality unless educators have the desire and put in the effort to make this a reality.

It is also possible that more forms of the introductory programming sequence emerge, beyond starting with the "CS1 for majors" and "CS1 for non-majors" that are common today. In fact, it is worthwhile considering the following question:

> *Should educators cling to the current and commonly accepted definitions [33] of CS1, CS2, and the introductory programming sequence in general, or should we abandon these widely clung-to and nearly ubiquitous notions altogether?*

Might we see the emergence of a new CS1, or a new family of introductory programming courses? Or will the possibilities and demands be such that the existing notions of introductory programming become so diluted that to think of CS1 as a foundation course – even with many different flavors – makes little sense? Might we see a suite of 'problem solving with programming' courses emerge each catering to different groups of learners in different ways? Could we actually see, much to the delight of former US president Barack Obama (and hopefully many others), programming taught just as much as reading (natural language), writing (natural language) and arithmetic[16]? Will we see programming as a *basic skill for all*[17] become a reality?

Will we see programming courses that don't require code writing, but code generation in ways that focus on ethics, usability, social implications, human factors, etc.? Skills that would be useful for any and all majors, whose applications may help more users? Might we see future computer science programs split into pathways (starting with their own introductory course, perhaps called "CS1" just for nostalgia) where the focus is on *generating* and understanding code (such as a CS1-A that requires understanding of code), and those that do not work with code at all (such as a CS1-B which leads to a path that focuses on non-code elements of computing, using AI to complete code-related tasks)? It becomes illustrative to envision different CS1-X courses taken by different groups of students. A small and most certainly non-exhaustive list of such courses follows:

1. *Software experts* who focus as professionals on developing complex software systems and computing tools (e.g., system software, IDEs, AI-powered tools), and who need to understand in depth what is happening 'under the hood'.
2. *Application developers* whose focus will be in developing applications for end users, who will use heavily AI tools to support coding, but for whom deep knowledge of what happens on the lower level is less relevant.
3. *IT specialists* whose main focus is on supporting customer interfaces, who need to know much about software but who do not develop software much – perhaps called conversational programmers.
4. *AI specialists* whose main focus is data analysis and processing with AI and ML tools and methods.
5. *Societal experts* whose work is analyzing the complex interactions and impacts that software applications have within the society. These professionals do not code but they need to understand what coding is and what can be done with coding and AI tools. These may be largely from disciplines outside computing.

---

[16] www.washingtonpost.com/news/the-switch/wp/2013/12/11/president-obama-talks-about-teaching-everyone-to-code-this-professor-does-it

[17] obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all

(6) *End users* from other disciplines who need to learn to use AI/ML tools to manage and process their data, to advance their disciplinary achievements and maximize utility and safety for disciplinary specialists from archaeology to zoology including but not limited to business, engineering, law, and medicine.

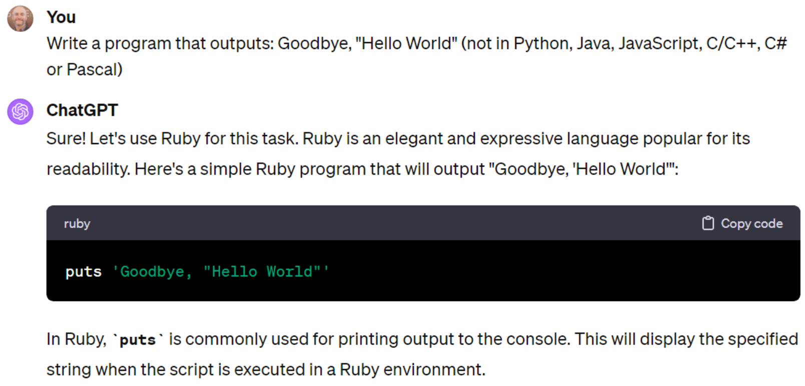
(7) *Creators* from all arts who create and leverage programmable (aka endlessly customizable) artifacts, products and services for artistic, entertainment, and enjoyment value of the public as well as experts in any and all manner of human artistic achievement.

Might we even see *the end of programming* [64]? Surely this would spell the end of CS1. However it would not mean the end of computing or computer science or the impact of programming on the world and society. It just might mean that to harness the power of computing, experts of all disciplines might not have to learn to type alphanumeric characters adhering to strict syntactic rules, starting in a class called CS1, in order to improve their work, art, disciplines and ultimately the lives of humans, and society at large.

9 CONCLUSION

It seems highly likely that introductory programming education will see substantial, if not foundational changes in the near future due to the influence of Generative AI on every aspect, from creating and changing learning goals through assessment, classroom practices, and tools. Generative AI also brings ethical and other concerns and may even reshape the wider computing curriculum and affect the very students that chose to study programming. It may, in the longer-term future, even affect who chooses to teach it.

Some of these changes may be welcome by many, particularly as CS1 has received what many consider to be disproportionate attention in terms of computing education research, despite the fact that in many ways too little has changed over the decades. Languages come and go but truly effective practices that gain large adoption such as pair programming, are rare. Even in 2023 many introductory programming courses start with writing code and never stop to reflect on other ways of engaging with, and learning about, programming. Many courses often start with students writing the same first program. Perhaps this is the beginning of a new era for programming education.



You

Write a program that outputs: Goodbye, "Hello World" (not in Python, Java, JavaScript, C/C++, C# or Pascal)

ChatGPT

Sure! Let's use Ruby for this task. Ruby is an elegant and expressive language popular for its readability. Here's a simple Ruby program that will output "Goodbye, 'Hello World'":

```ruby
puts 'Goodbye, "Hello World"'
```

In Ruby, `puts` is commonly used for printing output to the console. This will display the specified string when the script is executed in a Ruby environment.

Ultimately, the key to leveraging Generative AI may be less about what it can produce, and more about what it can do for students (and educators), which aligns with what the Artificial Intelligence in Education (AIEd) community has been discussing for years. However, for this to be the case, computing educators need to keep abreast of what Generative AI can and cannot do, and make a concentrated effort to leverage opportunity and benefit, and mitigate challenge and harm.

REFERENCES

[1] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 585–591. https://doi.org/10.1145/3287324.3287466

[2] A. Arora, M. Barrett, E. Lee, E. Oborn, and K. Prince. 2023. Risk and the future of AI: Algorithmic B, Data Colonialism, and Marginalization. *Information and Organization* 33, 3 (2023), 100478. https://doi.org/10.1016/j.infoandorg.2023.100478

[3] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 78 (apr 2023), 27 pages. https://doi.org/10.1145/3586030

[4] Brett Becker, James Prather, Paul Denny, Andrew Luxton-Reilly, James Finnie-Ansley, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation *(SIGCSE '23)*. ACM.

[5] Brett A. Becker. 2017. Artificial Intelligence in Education: What Is It, Where Is It Now, Where Is It Going. *Ireland's Yearbook of Education* 2018 (2017), 42–46.

[6] Brett A. Becker. 2019. Parlez-vous Java? Bonjour La Monde != Hello World: Barriers to Programming Language Acquisition for Non-Native English Speakers. In *30th Workshop of the Psychology of Programming Interest Group - PPIG '19* (Newcastle, UK). 13 pages. www.brettbecker.com/publications

[7] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research *(ITiCSE-WGR '19)*. ACM, NY, NY,USA, 177–210. https://doi.org/10.1145/3344429.3372508

[8] Brett A. Becker and Keith Quille. 2019. 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 338–344. https://doi.org/10.1145/3287324.3287432

[9] Jens Bennedsen and Michael E. Caspersen. 2019. Failure Rates in Introductory Programming: 12 Years Later. *ACM Inroads* 10, 2 (apr 2019), 30–36. https://doi.org/10.1145/3324888

[10] John Biggs. 1996. Enhancing Teaching Through Constructive Alignment. *Higher Education* 32, 3 (1996), 347–364.

[11] Benjamin S Bloom. 1984. The 2 sigma problem: The Search for Methods of Group Instruction as Effective as One-to-one Tutoring. *Educational researcher* 13, 6 (1984), 4–16.

[12] Dennis Bouvier, Ellie Lovellette, John Matta, Bedour Alshaigy, Brett A. Becker, Michelle Craig, Jana Jackova, Robert McCartney, Kate Sanders, and Mark Zarb. 2016. Novice Programmers and the Problem Description Effect. In *Proceedings of the 2016 ITiCSE Working Group Reports* (Arequipa, Peru) *(ITiCSE '16)*. ACM, NY, NY, USA, 103–118. https://doi.org/10.1145/3024906.3024912

[13] Vinton G. Cerf. 2023. Large Language Models. *Commun. ACM* 66, 8 (jul 2023), 7. https://doi.org/10.1145/3606337

[14] John Chen and Uri Wilensky. 2023. ChatLogo: A Large Language Model-Driven Hybrid Natural-Programming Language Interface for Agent-based Modeling and Programming. arXiv:2308.08102 [cs.HC]

[15] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* (2023), 111734.

[16] David De Cremer, Nicola Morini Bianzino, and Ben Falk. 2023. How Generative AI Could Disrupt Creative Work. *Harvard Business Review* 13 (2023).

[17] Paul Denny, Brett A Becker, Juho Leinonen, and James Prather. 2023. Chat Overflow: Artificially Intelligent Models for Computing Education renAIssance or apocAIypse? Keynote Address, 2023 ACM Conference on Innovation and Technology in Computer Science Education (ACM ITiCSE) Turku, Finland, July 12, 2023.

[18] Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. 2023. Can We Trust AI-Generated Educational Content? Comparative Analysis of Human and AI-Generated Learning Resources. arXiv:2306.10509 [cs.HC]

[19] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1136–1142. https://doi.org/10.1145/3545945.3569823

[20] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630909

[21] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2023. Promptly: Using Prompt Problems to Teach Learners How to Effectively Utilize AI Code Generators. arXiv:2307.16364 [cs.HC]

[22] Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. 2022. Robosourcing Educational Resources – Leveraging Large Language Models for Learnersourcing. https://doi.org/10.48550/ARXIV.2211.04715

[23] Heather Desaire, Aleesa E. Chua, Min-Gyu Kim, and David Hua. 2023. Accurately Detecting AI Text When ChatGPT is Told to Write Like a Chemist. *Cell Reports Physical Science* (2023), 101672. https://doi.org/10.1016/j.xcrp.2023.101672

[24] John Edwards, Joseph Ditton, Dragan Trninic, Hillary Swanson, Shelsey Sullivan, and Chad Mano. 2020. Syntax Exercises in CS1. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 216–226. https://doi.org/10.1145/3372782.3406259

[25] Ziv Epstein, Aaron Hertzmann, Investigators of Human Creativity, Memo Akten, Hany Farid, Jessica Fjeld, Morgan R Frank, Matthew Groh, Laura Herman, Neil Leach, et al. 2023. Art and the Science of Generative AI. *Science* 380, 6650 (2023), 1110–1111.

[26] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming *(ACE '22)*. ACM, Online, 10–19. https://doi.org/10.1145/3511861.3511863

[27] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises *(ACE '23)*. ACM, NY, NY, USA, 97–104. https://doi.org/10.1145/3576123.3576134

[28] Jingyuan Cai Keng Siau Fiona Fui-Hoon Nah, Ruilin Zheng and Langtao Chen. 2023. Generative AI and ChatGPT: Applications, Challenges, and AI-human Collaboration. *Journal of Information Technology Case and Application Research* 25, 3 (2023), 277–304. https://doi.org/10.1080/15228053.2023.2233814

[29] Joel E Fischer. 2023. Generative AI Considered Harmful. In *Proceedings of the 5th International Conference on Conversational User Interfaces* (Eindhoven, Netherlands) *(CUI '23)*. Association for Computing Machinery, New York, NY, USA, Article 7, 5 pages. https://doi.org/10.1145/3571884.3603756

[30] CC2020 Task Force. 2020. *Computing Curricula 2020: Paradigms for Global Computing Education.* Association for Computing Machinery, New York, NY, USA.

[31] Mark Guzdial. 2010. Does Contextualized Computing Education Help? *ACM Inroads* 1, 4 (dec 2010), 4–6. https://doi.org/10.1145/1869746.1869747

[32] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanp¨a¨a, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. *arXiv preprint arXiv:2306.05715* (2023).

[33] Matthew Hertz. 2010. What Do "CS1" and "CS2" Mean? Investigating Differences in the Early Courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 199–203. https://doi.org/10.1145/1734263.1734335

[34] Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. ChaTA: Towards an Intelligent Question-Answer Teaching Assistant using Open-Source LLMs. arXiv:2311.02775 [cs.LG]

[35] John Impagliazzo, Natalie Kiesler, Amruth N. Kumar, Bonnie Mackellar, Rajendra K. Raj, and Mihaela Sabin. 2022. Perspectives on Dispositions in Computing Competencies. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 662–663. https://doi.org/10.1145/3502717.3532121

[36] Bryn Jeffries, Jung A. Lee, and Irena Koprinska. 2022. 115 Ways Not to Say Hello, World! Syntax Errors Observed in a Large-Scale Online CS0 Python Course. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 337–343. https://doi.org/10.1145/3502718.3524809

[37] Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Peterson, and Sami Sarsa. 2022. Towards Giving Timely Formative Feedback and Hints to Novice Programmers. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. Association for Computing Machinery, New York, NY, USA, 95–115. https://doi.org/10.1145/3571785.3574124

[38] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J Cai, and Michael Terry. 2022. Discovering the Syntax and Strategies of Natural Language Programming with Generative Language Models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 386, 19 pages. https://doi.org/10.1145/3491102.3501870

[39] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[40] Natalie Kiesler. 2023. Beyond the Textbook: Rethinking Students' Competencies in the LLM Era. Generative AI: Implications for Teaching and Learning. https://doi.org/10.13140/RG.2.2.28355.37922/1

[41] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In *CoRR 2309.00029.* https://doi.org/10.48550/arXiv.2309.00029 arXiv:2309.00029 [cs.AI]

[42] Natalie Kiesler and Daniel Schiffner. 2023. Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. In *CoRR abs/2308.08572.* https://doi.org/10.48550/arXiv.2308.08572 arXiv:2308.08572 [cs.SE]

[43] KS Krishnapriya, Vítor Albiero, Kushal Vangara, Michael C King, and Kevin W Bowyer. 2020. Issues Related to Face Recognition Accuracy Varying Based on Race and Skin Tone. *IEEE Transactions on Technology and Society* 1, 1 (2020), 8–20.

[44] Sam Lau and Philip J. Guo. 2023. From "Ban It TillWe Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot *(ICER '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3568813.3600138

[45] Teemu Lehtinen, Lassi Haaranen, and Juho Leinonen. 2023. Automated Questionnaires About Students' JavaScript Programs: Towards Gauging Novice Programming Processes. In *Proceedings of the 25th Australasian Computing Education Conference.* 49–58.

[46] Teemu Lehtinen, Otto Seppälä, and Ari Korhonen. 2023. Automated Questions About Learners' Own Code Help to Detect Fragile Prerequisite Knowledge. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1.* 505–511.

[47] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 124–130. https://doi.org/10.1145/3587102.3588785

[48] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. arXiv:2308.06921 [cs.CY]

[49] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2018. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports* (Bologna, Italy) *(ITiCSE-WGR '17)*. Association for Computing Machinery, New York, NY, USA, 47–69. https://doi.org/10.1145/3174781.3174784

[50] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 931–937. https://doi.org/10.1145/3545945.3569785

[51] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 2022. Generating Diverse Code Explanations Using the GPT-3 Large Language Model *(ICER '22)*. ACM, NY NY, USA, 37–39. https://doi.org/10.1145/3501709.3544280

[52] Anne McDougall, John S Murnane, and Sandra Wills. 2014. The Educational Programming Language Logo: Its Nature and Its Use in Australia. *Reflections on the History of Computers in Education: Early Use of Computers and Teaching about Computing in Schools* (2014), 394–407.

[53] Robert K Merton. 1968. The Matthew Effect in Science: The Reward and Communication Systems of Science are Considered. *Science* 159, 3810 (1968), 56–63.

[54] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming With GitHub Copilot and ChatGPT.* Manning, Shelter Island, NY, USA. https://www.manning.com/books/learn-ai-assisted-python-programming

[55] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Peterson, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. https://doi.org/10.48550/arXiv.2310.00658 arXiv:2310.00658 [cs.CY] To appear in Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2023), July 7–12, 2023, Turku, Finland, 10.1145/3623762.3633499.

[56] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction* (Aug. 2023). https://doi.org/10.1145/3617367

[57] Rajendra Raj, Mihaela Sabin, John Impagliazzo, David Bowers, Mats Daniels, Felienne Hermans, Natalie Kiesler, Amruth N. Kumar, Bonnie MacKellar, Renée McCauley, Syed Waqar Nabi, and Michael Oudshoorn. 2021. Professional Competencies in Computing Education: Pedagogies and Assessment. In *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education* (Virtual Event, Germany) *(ITiCSE-WGR '21)*. Association for Computing Machinery, New York, NY, USA, 133–161. https://doi.org/10.1145/3502870.3506570

[58] Mihaela Sabin, Natalie Kiesler, Amruth N. Kumar, Bonnie K. MacKellar, Renee McCauley, Rajendra K. Raj, and John Impagliazzo. 2023. Fostering Dispositions and Engaging Computing Educators. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1216–1217. https://doi.org/10.1145/3545947.3569592

[59] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models *(ICER '22)*. ACM, NY NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[60] Simon, Andrew Luxton-Reilly, Vangel Ajanovski, Eric Fouh, Chris Gonsalvez, Juho Leinonen, Jack Parkinson, Matthew Poole, and Neena Thota. 2019. Pass Rates in STEM Disciplines Including Computing. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 267–268. https://doi.org/10.1145/3304221.3325532

[61] Pragnya Sridhar, Aidan Doyle, Arav Agarwal, Christopher Bogart, Jaromir Savelka, and Majd Sakr. 2023. Harnessing LLMs in Curricular Design: Using GPT-4 to Support Authoring of Learning Objectives. *arXiv preprint arXiv:2306.17459* (2023).

[62] Evrikleia Tsagala and Maria Kordaki. 2007. Critical Factors Influencing Secondary School Pupil's Decisions to Study Computing in Tertiary Education: Gender Differences. *Education and Information Technologies* 12 (2007), 281–295.

[63] Christopher Watson and Frederick W.B. Li. 2014. Failure Rates in Introductory Programming Revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (Uppsala, Sweden) *(ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 39–44. https://doi.org/10.1145/2591708.2591749

[64] Matt Welsh. 2022. The End of Programming. *Commun. ACM* 66, 1 (dec 2022), 34–35. https://doi.org/10.1145/3570220

[65] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 172–178. https://doi.org/10.1145/3545945.3569830

[66] Wenying Wu, Pavlos Protopapas, Zheng Yang, and Panagiotis Michalatos. 2020. Gender Classification and Bias Mitigation in Facial Images. In *Proceedings of the 12th ACM Conference on Web Science*. 106–114.

[67] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. 2022. Wordcraft: Story Writing With Large Language Models. In *27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22)*. Association for Computing Machinery, New York, NY, USA, 841–852. https://doi.org/10.1145/3490099.3511105