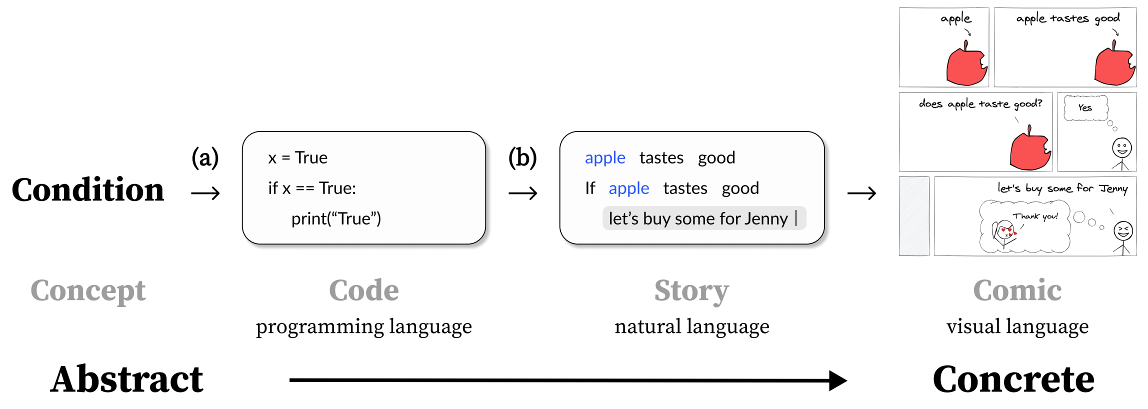
Leveraging Generative Conversational AI to Develop a Creative Learning Environment for Computational Thinking

SANGHO SUH, University of Waterloo, Canada
PENGCHENG AN, University of Waterloo, Canada



```
x = True
if x == True:
    print("True")
```

Fig. 1. CodeToon users co-create comics from code with generative conversational AI, which uses generative language models to generate (a) code examples for programming concepts and (b) stories from a given code.

We explore how generative conversational AI can assist students' learning, creative, and sensemaking process in a visual programming environment where users can create comics from code. The process of visualizing code in terms of comics involves mapping programming language (code) to natural language (story) and then to visual language (of comics). While this process requires users to brainstorm code examples, metaphors, and story ideas, the recent development in generative models introduces an exciting opportunity for learners to harness their creative superpower and researchers to advance our understanding of how generative conversational AI can augment our intelligence in creative learning contexts. We provide an overview of our system and discuss interaction scenarios to demonstrate ways we can partner with generative conversational AI in the context of learning computer programming.

CCS Concepts: • **Applied computing** → **Interactive learning environments**.

Additional Key Words and Phrases: visual programming environment, generative conversational AI, comics, coding strip

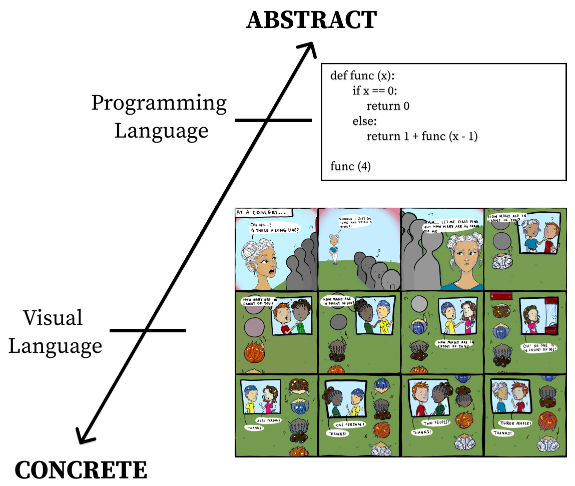
ACM Reference Format:
Sangho Suh and Pengcheng An. 2022. Leveraging Generative Conversational AI to Develop a Creative Learning Environment for Computational Thinking. In *27th International Conference on Intelligent User Interfaces (IUI '22 Companion), March 22–25, 2022, Helsinki, Finland*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3490100.3516473

1 INTRODUCTION

"Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction." - Jeannete Wing on Computational Thinking [7]

As technologies advance and play an increasingly larger role in our lives, computational thinking—the ability to understand computing concepts and procedures and their role in the tools we use—has become an important part of our training and education in the 21st century. Unfortunately, many find programming intimidating and difficult to learn because it requires learning concepts, languages, and procedures that have been abstracted through a series of abstractions. Programming concepts and languages employ abstract terms and unfamiliar syntax and conventions; yet how they relate to what we already know (e.g., real-life situations and equivalent/grounding metaphors) is not always explicated in learning materials or situations [6]. When learning programming procedures, how computer programs are executed and what happens (e.g., in memory) in each step are often omitted or presented as abstractions (e.g., loop), obscuring the process for novice learners who have not yet learned to trace (or abstract) execution steps.

If asked, the linguists S.I. Hayakawa and Wendell Johnson would point to 'dead-level abstracting' as a problem. In *Language in Thought and Action*, Hayakawa argues that moving between abstraction levels is critical to effective communication, saying that "*the informative speaker, the accurate thinker ... operate on all levels of the abstraction ladder, moving quickly and gracefully and in orderly fashion from higher to lower, from lower to higher*" [2]. *Dead-level abstracting* is a term referring to a linguistic phenomenon when a speaker is stuck in certain levels of abstraction. High or low, Hayakawa notes that this lack or absence of interplay between (concrete and abstract) abstraction levels makes it challenging to understand new information in a meaningful and effective way. Although many agree that the ability to "rapidly change levels of abstraction" is a key characteristic of computational thinking for computer scientists or engineers [1, 7], instructions in computing education tend to be mired in abstract levels of abstraction and lack opportunities for students to develop these skills.

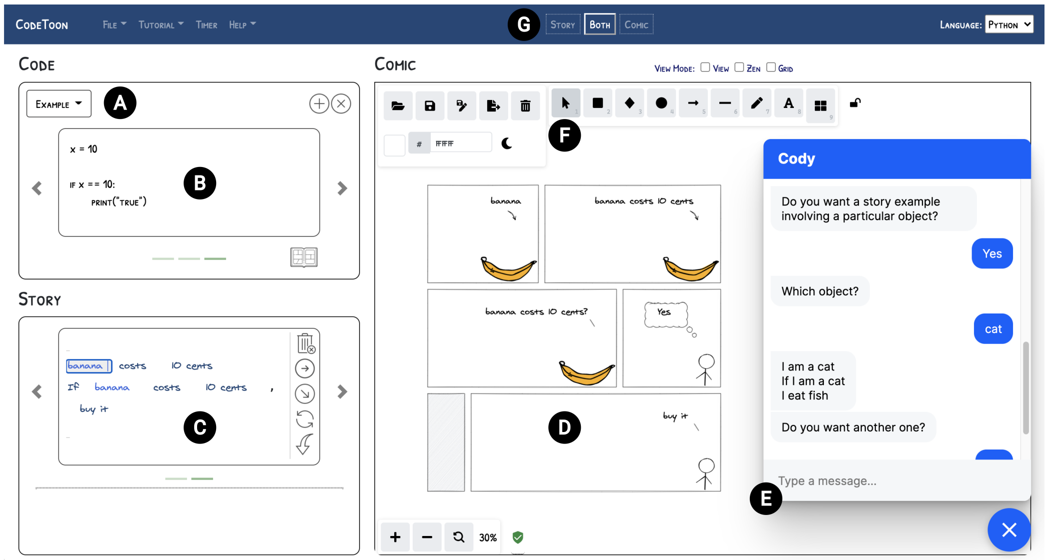

ABSTRACT
Programming Language
Visual Language
CONCRETE
AT A CONCERT...
OH NO...! IS THERE A LONG LINE?
SHOULD I JUST GO HOME AND WATCH A MOVIE?!
HMM... LET ME FIRST FIND OUT HOW MANY ARE IN FRONT OF ME
HOW MANY ARE IN FRONT OF YOU?
HOW MANY ARE IN FRONT OF YOU?
HOW MANY ARE IN FRONT OF YOU?
HOW MANY ARE IN FRONT OF YOU?
OH! NO ONE IS IN FRONT OF ME!
ZERO PERSONS!
THANKS!
ONE PERSON!
THANKS!
TWO PEOPLE!
THANKS!
THREE PEOPLE!
THANKS!

```
def func (x):
    if x == 0:
        return 0
    else:
        return 1 + func (x - 1)

func (4)
```

Fig. 2. Coding strip is different from comic books on programming as it is accompanied by its corresponding code, which allows readers to move up and down the ladder of abstraction [5].

Motivated by this observation, we formulated *coding strip* (Figure 2)—a form of comic strip (**concrete**) accompanied by its corresponding code (**abstract**)—to provide a model for learners to move between concrete and abstract levels of abstraction while learning programming concepts, languages, and procedures. In our prior work, we explored the design process for creating coding strips with teachers and students; we found that the process of creating comics from programming concepts allows both teachers and students to engage in a fun, creative learning process [6]. This demo showcases CodeToon, a digital tool developed to facilitate this design process. It extends creativity support tools from prior work with generative conversational AI (CAI). Specifically, while teachers and students had to come up with their own code and stories in prior work, they can now use digital drawing tools and generative CAI to generate code and stories for them.



```
x = 10

if x == 10:
    print("true")
```

banana costs 10 cents
If banana costs 10 cents , buy it

Do you want a story example involving a particular object?

Yes

Which object?

cat

I am a cat
If I am a cat
I eat fish

Do you want another one?

Type a message...

Fig. 3. CodeToon interface: (A) drop-down allows users to add code examples for basic programming concepts; (B) code container; (C) story template container where users can add stories that correspond to the code snippet; (D) drawing canvas for comics; (E) generative conversational AI that users can interact with; (F) styles and tools panels; and (G) buttons for changing the interface layout (between code & story, current, or canvas-only layout).

2 CODETOON: VISUAL PROGRAMMING ENVIRONMENT

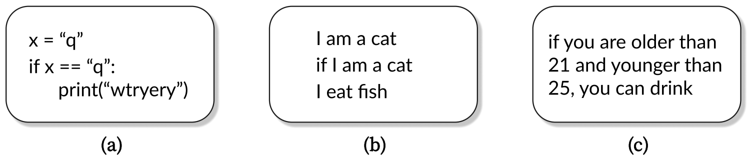
CodeToon (Figure 3) is implemented using Bootstrap and Django. The drawing canvas is implemented using an open-source tool, Excalidraw. CodeToon provides users unhindered experience on both desktop and tablet. They can use the system in either form factor without missing any features or interactions. For instance, they can use mouse, touch, or stylus (e.g., Apple pencil, Wacom stylus) for drawing and editing comics on the canvas.

2.1 Interaction Scenarios & Examples

In this section, we contextualize our design by providing two usage scenarios that show how learners can partner with the generative CAI to get creative support in creating code and stories based on computational concepts. In addition, we also provide real examples of code and stories generated by our utilized generative model, to demonstrate what kind of outcomes users can expect from the generative CAI.

2.1.1 Scenario A: Co-creating Code. Tom is a learner who uses CodeToon to learn basic programming concepts. He wants to co-create a comic with CodeToon that could help him and his friend concretely understand the concept of 'condition.' As the first step, he wants to get some inspiration about what kind of code snippets he could base on to create a story. So he opens the chat window of the conversational AI and types: "I need help." The AI asks "what do you need help with?" and displays two options: 'Code' and 'Story.' Tom presses 'Code,' and the AI replies, "Sure! Do you want a code example involving a particular concept?" Tom chooses 'Yes' and types 'condition.' The AI then shows an example to Tom. Tom wants more options, so he asks the AI to give a few more examples. In the end, he decides to use the third example. He clicks the code to paste it into the code container (Figure 3 (B)) and further modifies it to finalize his code snippet.

2.1.2 Scenario B: Co-Creating Story. After Tom finishes modifying his code snippet, he needs to turn the abstract code into a concrete, fun narrative that can help him and his friend learn and remember the concept. Again he needs some inspiration so he asks the CAI for ideas. After Tom requests help with the story ideation, the CAI asks "do you want the story to include a particular object or character?" Tom thinks it would be fun to have a cat in the story because he and his friend love cat. So he types in 'cat.' And the AI asks "do you want me to create a story based on your code in the code block?" Tom confirms and the CAI generates a story. Tom continues asking the AI to generate a few more stories until he finds a funny one he is satisfied with. He clicks the story to automatically copy it into the story block. In the story block (Figure 3 (C)), he can still modify text fields in the story (e.g., the name of the characters or events).


```
x = "q"
if x == "q":
    print("wtryery")
```

I am a cat
if I am a cat
I eat fish

if you are older than
21 and younger than
25, you can drink

Fig. 4. Output examples: (a) code generated by the model; (b & c) stories generated by the model based on given code snippets.

2.1.3 Examples of Generated Code and Stories. CodeToon uses a GPT-3 model to generate code examples and stories. Namely, if users need code examples for 'condition,' in the back-end, the system will feed the generative model a prompt to indicate that it should generate a code snippet with 'condition.' Figure 4 (a) shows a code example for 'condition' generated by the model. As can be seen, we designed our prompt to make the model generate a simple code snippet so that users can add additional code or modify the generated code. This way, we motivate co-creation between users and the AI rather than automate all the work and leave no room for users to experiment and exercise their creativity. Similarly, as shown by Figure 4 (b & c), the stories generated by the model based on given code snippets are also simple, which makes it more open-ended for user modifications. As Figure 4 (c) shows, the story created by generative models can make some funny mistakes, e.g., inaccurate descriptions or off-putting statements that may diverge from common sense. However, we see this more as an opportunity for serendipitous discovery, encouraging out-of-the-box ideas, and motivating users to be active participants in this co-creative process [3, 4].

3 CONCLUSION AND FUTURE WORK

We presented CodeToon, a visual programming environment that invites users to co-create comics from code with generative conversational AI. This work is exciting for several reasons. First, while this work focuses on programming, other technical areas such as math, engineering, and AI can benefit from this work as they also experience the same challenges of teaching abstract concepts, languages, and procedures. Once this research matures and we can leverage generative models to automatically create abstraction layers, it can significantly improve how we interact with information, as we can use the right level(s) of abstraction for the situation. Secondly, this work contributes a concrete setting and system for us to explore questions such as how the user needs will drive the development of generative AI algorithms and how these models can create compelling co-creative experiences. While we described two interaction scenarios, there are many other ways to design the prompt. Thus we should investigate how different prompt designs influence co-creative experiences. While CodeToon offers templates for the quick and easy creation of comics, it does not yet leverage generative models to create vector-based images and add to the comic canvas. As generative models are capable of creating drawings, this may be an interesting feature to add next, to advance the authoring experience.

REFERENCES

[1] Juris Hartmanis. 1994. Turing award lecture: On computational complexity and the nature of computer science. *Commun. ACM* 37, 10 (1994), 37–44.

[2] Samuel Ichiyé Hayakawa and Alan R Hayakawa. 1990. *Language in thought and action*. Houghton Mifflin Harcourt.

[3] Ronald S Lenox. 1985. Educating for the serendipitous discovery. *Journal of Chemical Education* 62, 4 (1985), 282.

[4] Lori McCay-Peet and Elaine G Toms. 2010. The process of serendipity in knowledge work. In *Proceedings of the third symposium on Information interaction in context*. 377–382.

[5] Sangho Suh. 2020. Promoting Meaningful Learning by Supporting Interplay within Abstraction Ladder. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–2.

[6] Sangho Suh, Martinet Lee, Gracie Xia, et al. 2020. Coding strip: A pedagogical tool for teaching and learning programming concepts through comics. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–10.

[7] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.