Alifmatika: Jurnal Pendidikan dan Pembelajaran Matematika
Volume 7, Issue 1, 34-52, June 2025
e-ISSN: 2715-6109 | p-ISSN: 2715-6095
https://journal.ibrahimy.ac.id/index.php/Alifmatika

A learning trajectory for developing computational thinking in prospective mathematics teachers through Python programming in Google Colab

Edi Irawan[1*], Moh. Khoridatul Huda[2], Ratni Purwasih[3]

[1*]Tadris Matematika, Universitas Islam Negeri Kiai Ageng Muhammad Besari Ponorogo, East Java 63471, Indonesia
[2]Pendidikan Guru Madrasah Ibtidaiyah, Universitas Islam Raden Rahmat, East Java 65163, Indonesia
[3]Pendidikan Guru Sekolah Dasar, Institut Keguruan dan Ilmu Pendidikan (IKIP) Siliwangi, West Java 40521, Indonesia

[1*]nawariide@iainponorogo.ac.id, [2]moh.huda@uniramalang.ac.id, [3]ratnipurwasih@ikipsiliwangi.ac.id

Received: March 22, 2025 | Revised: April 28, 2025 | Accepted: May 15, 2025 | Published: June 15, 2025
*Corresponding author

Abstract:
Computational thinking (CT) is a fundamental skill that needs to be developed by prospective mathematics teachers to improve problem-solving and logical reasoning. Integrating programming into mathematics learning is an effective approach to training this skill. This study aimed to design a hypothetical learning trajectory (HLT) for developing CT using Python programming on Google Colab. This study used a didactical design research (DDR) framework consisting of three stages: prospective analysis, metapedadidactic analysis, and retrospective analysis. The research participants were prospective mathematics teacher students enrolled in a computer programming course. Data were collected through observation, code artefacts, and reflective interviews. The results showed that HLT, designed in stages, improved the four main components of CT: decomposition, abstraction, pattern recognition, and algorithmic thinking. The students experienced improvements in breaking down problems, devising more efficient solutions, recognising patterns in code structures, and systematically designing algorithms. In addition, Google Colab supports learning by providing a collaborative and accessible programming environment. However, minor syntax errors and lack of attention to indentation were found. This study recommends using structured debugging strategies and project-based learning in optimizing CT development. The findings indicate that the integration of programming into the education of prospective mathematics teachers can equip them with essential CT skills to support technology-based mathematics teaching.

Keywords: Computational Thinking; Google Colab; Hypothetical Learning; Learning Trajectory; Prospective Math Teacher; Python Programming.

How to Cite: Irawan, E., Huda, M. K., & Purwasih, R. (2025). A learning trajectory for developing computational thinking in prospective mathematics teachers through Python programming in Google Colab. *Alifmatika: Jurnal Pendidikan dan Pembelajaran Matematika*, 7(1), 34-52. https://doi.org/10.35316/alifmatika.2025.v7i1.34-52

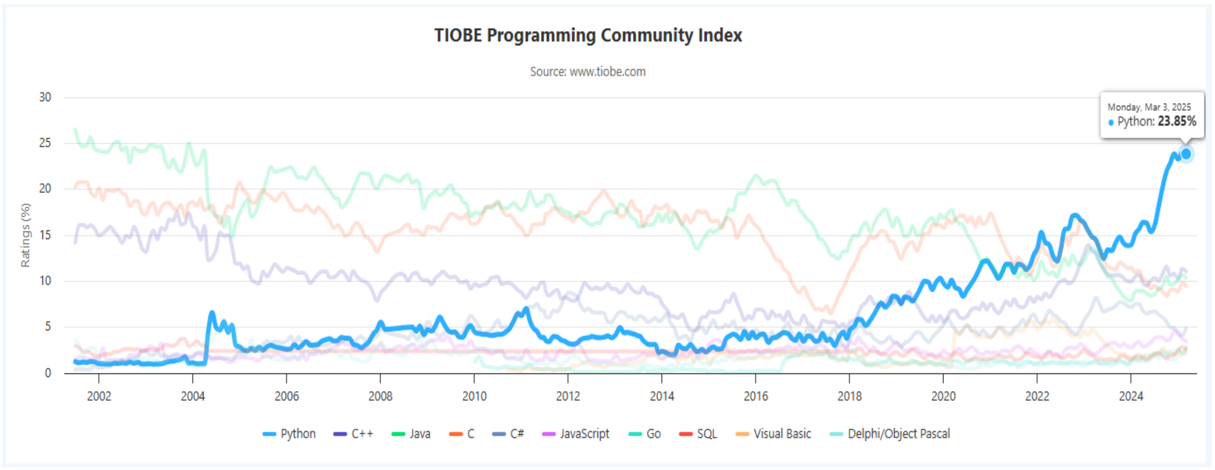
Introduction

Integrating computational thinking (CT) into mathematics education has become a global concern because of its role in developing complex problem-solving skills. According to Wing (2006, 2017), CT involves abstraction, decomposition, pattern recognition, and algorithmic thinking, all essential in solving complex mathematical problems. In Indonesia, the Ministry of Education has emphasised the importance of digital literacy and CT in the curriculum in an effort to produce a highly competitive generation (Badan Standar, Kurikulum, dan Asesmen Pendidikan Kementerian Pendidikan Dasar dan Menengah Republik Indonesia, 2025). Mathematics is a subject highly connected and relevant to CT (Irawan et al., 2024b; Irawan & Herman, 2023). Therefore, prospective mathematics teachers also need to be equipped with CT skills in order to be able to solve problems systematically and effectively.

A mathematician, Seymour Papert (1980, 1996) first introduced CT. However, the term CT developed rapidly after it was redefined by Wing (2006) as a way of thinking that enables one to formulate and solve problems systematically with computational principles. Since then, the study of CT has undergone significant developments (Ilic et al., 2018; Tekdal, 2021). CT is the thought process involved in formulating a problem so that its solution can be represented as computational steps and algorithms (Aho, 2012). CT is recognised as a fundamental skill that can be applied to various disciplines, including mathematics and science (Grover & Pea, 2018). CT skills involve coding and logical, analytical, and systematic thinking patterns in solving problems effectively (Shute et al., 2017). CT is a combination of logical thinking, abstraction, and automation, whereas others emphasise aspects such as problem decomposition, data representation, and debugging (Weintrop et al., 2015). Therefore, CT is one of the essential competencies in education in the 21st century.

In addition to experiencing developments in the definition aspect, CT has also experienced developments in its aspects or components. One of the practical and widely used groupings of CT components is decomposition, abstraction, pattern recognition, and algorithms (Badan Standar, Kurikulum, dan Asesmen Pendidikan Kementerian Pendidikan Dasar dan Menengah Republik Indonesia, 2025; Dong et al., 2019). Decomposition emphasizes breaking down large problems into smaller parts that are easier to manage and solve (Dong et al., 2019; Palts & Pedaste, 2020; Wing, 2017). Pattern recognition helps in identifying similarities in data or problems so that it can be used to find similar solutions in other situations (Angeli & Giannakos, 2020; Dong et al., 2019). Abstraction emphasizes more on filtering out important and necessary information in solving a problem (Cansu & Cansu, 2019; Dong et al., 2019; Wing, 2006). Finally, algorithmic thinking focuses on composing logical and systematic steps in solving a problem (Cansu & Cansu, 2019; Dong et al., 2019; Wing, 2006). In this study, these four components are targeted to be developed in learning Python programming for prospective mathematics teachers.

Researchers have explored various methods for integrating CT into mathematics education. Various studies have reported that programming is one of the most promising approaches and has proven effective in developing CT (Kong et al., 2020; Sun & Zhou, 2023, 2023; Wei et al., 2021). Python is a programming program that is widely used in research to develop CT (Irawan et al., 2024a). Many studies have reported that learning Python programming is very effective for honing and developing students' CT (Bai et al., 2021, 2021; Choi & Choi, 2024; De Jesús & Martinez, 2020; Hsiao et al., 2023; Jesús & Martinez, 2023; Kamak & Mago, 2023; Kim et al., 2019; Ren et al., 2021; Saha, 2015). Python is currently the most widely used programming language (Jansen, 2025), as shown in Picture 1. Although many previous studies have shown the benefits of Python programming in developing CT, studies focused on sharpening the CT of prospective mathematics teachers are limited.



Picture 1. Trend use of Python programming

Preliminary studies have shown that not all prospective mathematics teachers have supportive devices. One potential solution to address this gap is to utilize Google Colab, a cloud-based platform that enables real-time collaboration and interactive programming (Naik et al., 2021; Vallejo et al., 2022). Google Colab offers advantages such as seamless access to computing resources and an integrated development environment for Python (Ferreira et al., 2024; Kuroki, 2021; Llerena-Izquierdo et al., 2024; Naik et al., 2021). These features make Google Colab suitable for developing CT for prospective mathematics teachers. However, there are still no efforts to develop CT for prospective mathematics teachers through developing a hypothetical learning trajectory (HLT) in programming courses using Python on Google Colab.

The HLT is a teacher's prediction of the path that students can take when learning mathematics (Clements & Sarama, 2024; Simon, 2020). Furthermore, Simon (1995, 2020) mentioned that HLT consists of three main components: learning goals, learning activities, and the hypothesis of the learning process. There are five stages in the preparation of HLT, namely: (1) extracting learning goals (LG) from the literature; (2) categorizing learning goals; (3) clustering learning goals; (4) assembling clusters into trajectories; and (5) assigning levels of evidence for goals and relationships (Rich et al., 2017). In this study, we defined HLT as a storyline about teaching and learning Python programming in Google Colab to hone the CT skills of prospective math teachers. The storyline includes three interrelated aspects: (1) learning objectives regarding students' CT, (2) a sequence of instructional tasks that engage students, and (3) a series of tasks that guide students' mathematical activities.

To bridge this gap, this study aims to develop a Python programming HLT on Google Colab to develop the CT of prospective mathematics teacher students. HLT provides a structured approach to carefully designing learning activities to achieve learning goals by considering students' initial abilities (Clements & Sarama, 2024; Simon, 2020). In addition to mastering the basic concepts of Python to support mathematics learning, this study also sought to hone students' CT skills. Therefore, this study aims to contribute to the development of effective didactic strategies for integrating CT into mathematics education.

Research Methods

This study used a qualitative approach with an interpretative critical paradigm. The critical paradigm was used in the development of HLT oriented to hone students' CT skills through the use of Python programming. The interpretative paradigm was used to interpret data related to the development of students' CT, both in terms of decomposition, pattern recognition, abstraction, and algorithms. This study uses the didactic design research (DDR) framework. DDR emphasizes the balance of knowledge diffusion and acquisition processes (Suryadi, 2013).

Procedures

The procedure in this study followed the three stages of the DDR framework (Suryadi, 2019). First, at the prospective analysis stage, the initial HLT was developed based on literature review, expert consultation, and analysis of the knowledge and needs of prospective mathematics teachers. HLT was organised into a series of activities that progressively introduced CT concepts through Python programming on Google Colab. Second, in the metapedadidactic analysis stage, teaching was conducted to implement the previously designed HLT in the classroom. HLT implementation was conducted in four lecture meetings, with the materials and tasks listed in the HLT. Third, at the retrospective analysis stage, qualitative data were analysed to evaluate the effectiveness of HLT in honing students' CT skills. The analysis focused on the impact of Python programming on their CT skills and the learning barriers faced during the learning process.

Participants

This study involved prospective mathematics teacher students at IAIN Ponorogo who were taking a computer programming course. A total of 46 students took this course, consisting of 6 male and 40 female students. The results of the initial identification show that as many as one person has learned C++ programming, and four people have learned basic visual programming in Microsoft Excel. However, none of the participants had previously learned Python. Owing to ethical considerations, the participants' identities in this study were anonymous.

Data Collection

The data collected in this study included three factors. First, observational data during the HLT implementation process were collected during four lecture meetings. Second, data in the form of artifacts are in the form of Python coding in Google Colab, which is well documented in Google Drive. Third, data related to the development of CT students in terms of decomposition, pattern recognition, abstraction, and algorithms. These three data sets support each other and complement the material in the qualitative data analysis.

Data Analysis

The qualitative data, including interview transcripts and field notes, were subsequently analyzed using Atlas.ti® version 9 software. The analysis process was conducted in three stages: open, axial, and selective coding (Corbin & Strauss, 2015). In the open coding stage, all Python coding artifacts and interview results were coded to identify the keywords or main concepts that appeared in the data. Next, at the axial coding stage, the previously created codes were grouped into categories to produce meaningful patterns. The final stage, selective coding, was carried out by integrating and refining the categories and patterns formed from the overall categories of the data. Through these three stages, findings can be produced that provide a comprehensive picture of the development of students' CT skills.

Data Validation

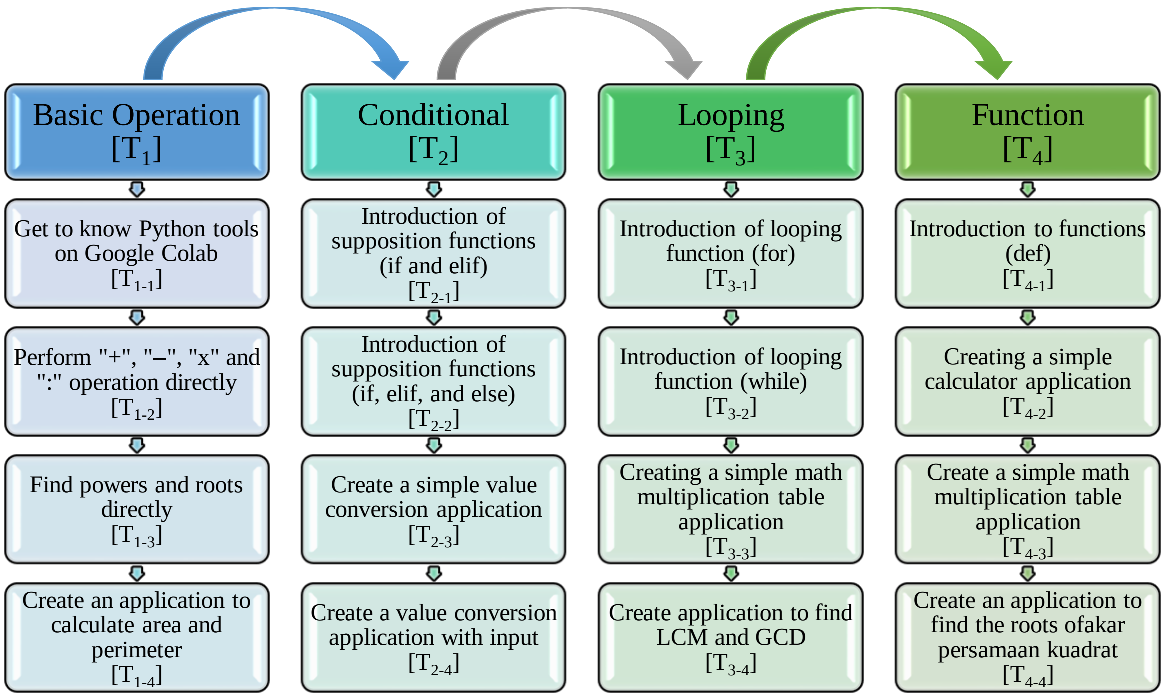
The validity of the data in this study included four aspects: transferability, credibility, dependability, and confirmation (Denzin & Lincoln, 2018). Transferability is achieved by providing a detailed description of the research context, research procedures, learning procedures, and research findings. Credibility was maintained through the triangulation of sources and methods by comparing the results of observations, artifact analysis, and interviews, as well as data analysis using the Atlas.ti software. Reliability was ensured by conducting an audit trail, which provided detailed documentation of each stage of the research, from data collection, data analysis, interpretation of results, and documentation of the coding process using Atlas.ti. Finally, confirmability was achieved by double-checking each step of the analysis and discussing the findings with peers or experts to gain a more objective perspective.

Results and Discussions

In accordance with the research framework used, the findings and discussion of this study are presented in three phases of DDR research: prospective analysis, metapedadidactic analysis, and retrospective analysis.

Prospective Analysis

In the prospective analysis stage, a series of activities was conducted to develop the HLT. As mentioned by Simon (1995, 2020), HLT consists of three components: learning goals, learning activities, and the hypothesis of the learning process. Therefore, the first step was to determine learning goals. The learning goals were based on identifying the students' initial understanding of coding using Python programming. As a result, none of the participating students had learned Python independently or classically at the previous level. On the other hand, the computer programming course equips prospective mathematics teachers with foundational programming skills that can enhance mathematics instruction in their future classrooms. Therefore, the learning goal in this research is that students can create simple mathematical applications by utilising Python's basic functions while honing their CT skills. This study focuses on four basic functions in Python programming: arithmetic operations, conditions, looping, and functions. In accordance with the objectives and scope of this study, we developed an HLT, as presented in Picture 2.


Basic Operation [T_1]
Get to know Python tools on Google Colab [T_{1-1}]
Perform "+", "–", "x" and ":" operation directly [T_{1-2}]
Find powers and roots directly [T_{1-3}]
Create an application to calculate area and perimeter [T_{1-4}]
Conditional [T_2]
Introduction of supposition functions (if and elif) [T_{2-1}]
Introduction of supposition functions (if, elif, and else) [T_{2-2}]
Create a simple value conversion application [T_{2-3}]
Create a value conversion application with input [T_{2-4}]
Looping [T_3]
Introduction of looping function (for) [T_{3-1}]
Introduction of looping function (while) [T_{3-2}]
Creating a simple math multiplication table application [T_{3-3}]
Create application to find LCM and GCD [T_{3-4}]
Function [T_4]
Introduction to functions (def) [T_{4-1}]
Creating a simple calculator application [T_{4-2}]
Create a simple math multiplication table application [T_{4-3}]
Create an application to find the roots ofakar persamaan kuadrat [T_{4-4}]

Picture 2. Hypothetical learning trajectory programming course to promote CT

Picture 2 shows the HLT, especially in the second and third components, namely the learning activity and the hypothesis of the learning process. The learning process of programming courses with Python starts with Task 1 (basic operation), Task 2 (conditional), Task 3 (looping), and finally Task 4 (function). Task 1 begins with an introduction to cloud-based Python programming on Google Colab, performs addition, subtraction, multiplication, division, power, and root operations directly, and is challenged to create a simple application to find the area and perimeter. Task 2 (condition) starts with the introduction of the supposition function ("if" and "elif"), then ("if", "elif", and "else"). Students are challenged to create a simple application for value conversion directly and with input using "if, elif, and else". Task 3 (looping) starts with the introduction and elaboration of the looping functions (for) and (while), then students are challenged to create an application to create a mathematical multiplication table. Finally, Task 4 (functions) starts with an introduction to functions ("def"). Students are challenged to create a simple calculator application, a simple value conversion application, and an application to find the roots of a quadratic equation using Python's "def" function.

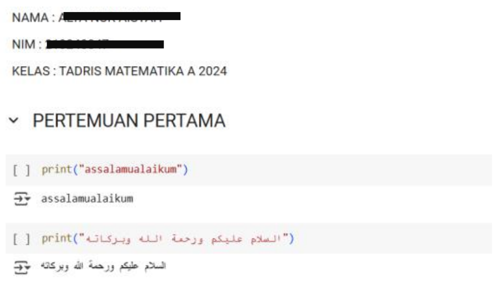
Another goal of HLT development was to hone prospective mathematics teachers' CT skills. Four CT skills were gradually honed in each task. The details of the activities to hone CT through Python programming courses are presented in Table 1. Table 1 shows that the development of HLT considers not only the conceptual stages in Python programming but also explicit strategies for honing the CT skills of prospective mathematics teachers. This HLT development is in accordance with the concept of HLT development, where learning objectives are based on students' initial abilities considering the stages of students' thinking development (Simon, Kara, et al., 2018; Simon, Placa, et al., 2018). Based on the initial abilities possessed by students, to achieve the set goals, a series of tasks are arranged, starting from Tasks 1, 2, 3, and 4. This task arrangement is done coherently, relevant to students' thinking, and in accordance with their learning objectives (Clements & Sarama, 2024). Similarly, the preparation of the stages of the learning process in each task, from the simplest basic concepts to those that can be applied in everyday life, especially those related to mathematics. Therefore, through this HLT, the learning process is based on predictions of how students' thinking and understanding will develop, not just trial and error (Simon, 1995; Simon & Tzur, 2004). The various Python programming projects given are also strongly related to the mathematical context (Guillod, 2024; Saha, 2015), making them very relevant for prospective mathematics teachers.

Table 1. Promoting CT's use of Python programming

No.	Series of Task	Decomposition	Abstraction	Pattern Recognition	Algorithm
1.	Task 1 (Basic Operation)	Students break down the problem into basic mathematical operations before combining it into a single program.	Identify the important parts of the problem (e.g. only the side lengths are required to calculate the area).	Discover patterns of mathematical operations that are often used in programming.	Calculate the steps in a logical sequence using Python syntax.
2.	Task 2 (Conditional)	The conditions are separated into various possible outputs based on user input.	Specify important conditions that should be tested in a program, such as threshold values.	Recognise decision patterns based on given conditions.	Compose the decision steps systematically using an if-elif-else structure.
3.	Task 3 (Looping)	Breaks repetitive tasks into small steps that can be automated.	Select relevant information in the looping patterns to avoid manually repeating the code.	Recognise patterns in iterative processes, such as multiplication tables.	Constructing an efficient looping logic such that the program runs automatically with the correct number of iterations.
4.	Task 4 (Function – def)	Large problems are divided into smaller functions that are more modular and reusable.	Understanding the role of parameters and return values in simplifying the code.	Recognise patterns of function usage in various programming contexts.	Construct function-based algorithms to make the code more efficient and organised.

Metapedadidactic Analysis

This metapedadidactic analysis stage focused on observing how the HLT that had been designed could be implemented directly. As previously designed, HLT was implemented in four lecture meetings. In the first meeting, students were introduced to the Python programming language as a simple programming language (Downey, 2024; Guillod, 2024; Zhuang et al., 2025) and the number of users increased even more in March 2025 (Jansen, 2025). Because not all the students had computers or laptops, the Python learning process was performed using Google Colab. Google Colab is a platform that provides Python online at https://colab.research.google.com, and integrates it directly with Google Drive (Naik et al., 2021). Thus, the entire Python coding process in Google Colab is automatically stored on Google Drive. Additionally, the resulting Python coding can be easily shared or emailed to others (including lecturers). An example of student-generated coding artefacts in Task 1 is shown in Picture 3.



NAMA : [redacted]

NIM : [redacted]

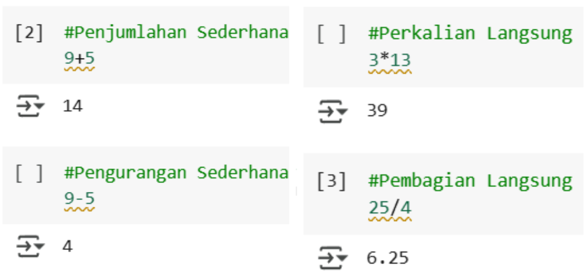
KELAS : TADRIS MATEMATIKA A 2024

PERTEMUAN PERTAMA

```
[ ] print("assalamualaikum")
assalamualaikum

[ ] print("السلام عليكم ورحمة الله وبركاته")
السلام عليكم ورحمة الله وبركاته
```

(a) Artifact examples on T1-1



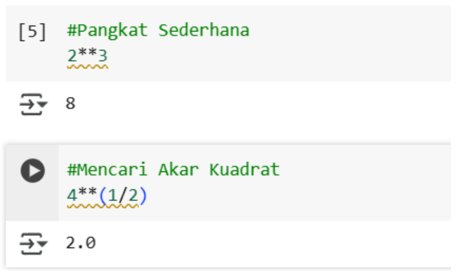
```
[2] #Penjumlahan Sederhana
    9+5
14

[ ] #Perkalian Langsung
    3*13
39

[ ] #Pengurangan Sederhana
    9-5
4

[3] #Pembagian Langsung
    25/4
6.25
```

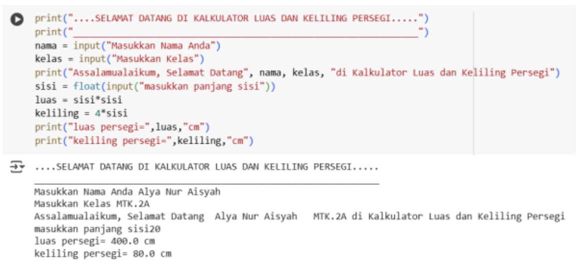
(b) Artifact examples on T1-2



```
[5] #Pangkat Sederhana
    2**3
8

    #Mencari Akar Kuadrat
    4**(1/2)
2.0
```

(c) Artifact examples on T1-3



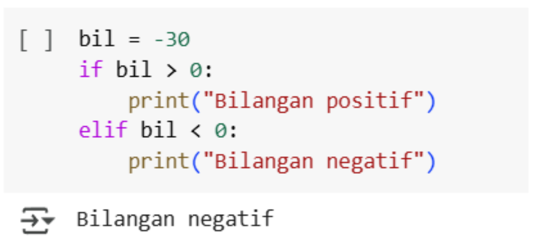
```
print("....SELAMAT DATANG DI KALKULATOR LUAS DAN KELILING PERSEGI.....")
print("______________________________________________________________")
nama = input("Masukkan Nama Anda")
kelas = input("Masukkan Kelas")
print("Assalamualaikum, Selamat Datang", nama, kelas, "di Kalkulator Luas dan Keliling Persegi")
sisi = float(input("masukkan panjang sisi"))
luas = sisi*sisi
keliling = 4*sisi
print("luas persegi=",luas,"cm")
print("keliling persegi=",keliling,"cm")

....SELAMAT DATANG DI KALKULATOR LUAS DAN KELILING PERSEGI.....
______________________________________________________________
Masukkan Nama Anda Alya Nur Aisyah
Masukkan Kelas MTK.2A
Assalamualaikum, Selamat Datang  Alya Nur Aisyah   MTK.2A di Kalkulator Luas dan Keliling Persegi
masukkan panjang sisi20
luas persegi= 400.0 cm
keliling persegi= 80.0 cm
```

(d) Artifact examples on T1-4

Picture 3. Coding artifact example on Task 1

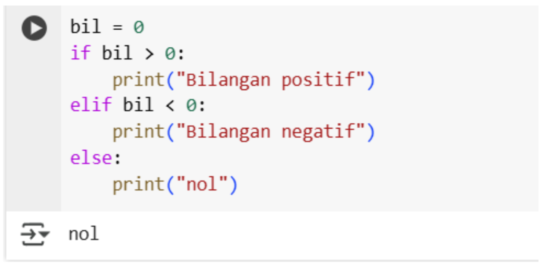
Picture 3 shows various evidence of coding artefacts performed by students related to the utilisation of various basic functions in Python. Students can use several functions, such as displaying output using the "print" function and performing simple mathematical operations, including addition, subtraction, multiplication, division, power, and root. Students also managed to create a simple application to find the area of a square (T1-4), where the user can input the side length using the "input" function. The application will display the area and perimeter in Picture 3(d). The results of the observations and interviews with respondents showed no problems in completing Task 1. Furthermore, in the second meeting, students learned the use of conditional functions ("if", "elif", and "else") and applied them in the context of daily life problems, as presented in Picture 4.



```
[ ] bil = -30
    if bil > 0:
        print("Bilangan positif")
    elif bil < 0:
        print("Bilangan negatif")

Bilangan negatif
```

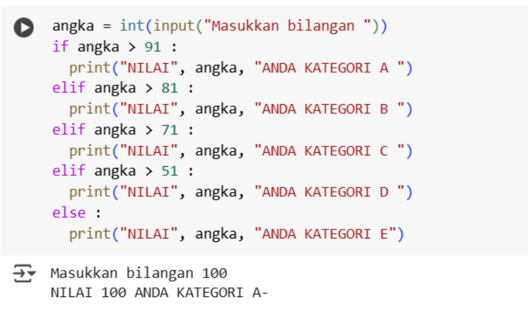
(a) Artifact examples on T2-1



```
bil = 0
if bil > 0:
    print("Bilangan positif")
elif bil < 0:
    print("Bilangan negatif")
else:
    print("nol")

nol
```

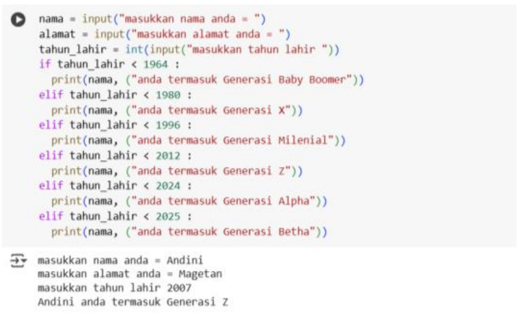
(b) Artifact examples on T2-2



```
angka = int(input("Masukkan bilangan "))
if angka > 91 :
  print("NILAI", angka, "ANDA KATEGORI A ")
elif angka > 81 :
  print("NILAI", angka, "ANDA KATEGORI B ")
elif angka > 71 :
  print("NILAI", angka, "ANDA KATEGORI C ")
elif angka > 51 :
  print("NILAI", angka, "ANDA KATEGORI D ")
else :
  print("NILAI", angka, "ANDA KATEGORI E")
```

```
Masukkan bilangan 100
NILAI 100 ANDA KATEGORI A-
```

(c) Artifact examples on T_{2-3}



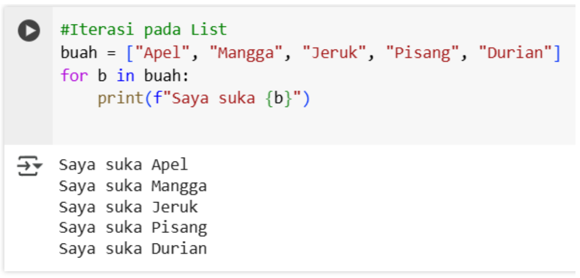
```
nama = input("masukkan nama anda = ")
alamat = input("masukkan alamat anda = ")
tahun_lahir = int(input("masukkan tahun lahir "))
if tahun_lahir < 1964 :
  print(nama, ("anda termasuk Generasi Baby Boomer"))
elif tahun_lahir < 1980 :
  print(nama, ("anda termasuk Generasi X"))
elif tahun_lahir < 1996 :
  print(nama, ("anda termasuk Generasi Milenial"))
elif tahun_lahir < 2012 :
  print(nama, ("anda termasuk Generasi Z"))
elif tahun_lahir < 2024 :
  print(nama, ("anda termasuk Generasi Alpha"))
elif tahun_lahir < 2025 :
  print(nama, ("anda termasuk Generasi Betha"))
```

```
masukkan nama anda = Andini
masukkan alamat anda = Magetan
masukkan tahun lahir 2007
Andini anda termasuk Generasi Z
```

(d) Artifact examples on T_{2-4}

Picture 4. Coding artifact example on Task 2

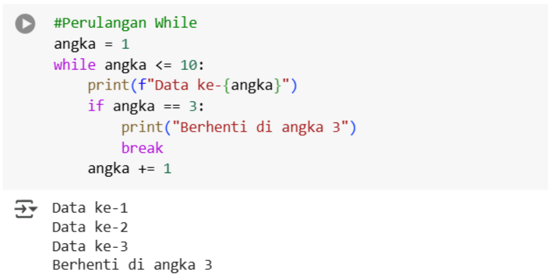
Picture 4 shows an example of using conditional functions in stages, starting from the simplest stage. In the early stage (T_{2-1}), students try to use only "if" and "elif" functions to determine whether a number is positive or negative. Next, they added the "else" function to add another category (T_{2-3}). Next, the students created a simple value conversion application (T_{2-3}), where the user can enter any value using the "input" function. This challenge resulted in the application of value conversion, as shown in Pciture 4(c). In the final session (T_{2-4}), students created an application to detect generation categories (baby boomer, X, millennial, Z, alpha, or beta), where users can enter their name, address, and year of birth using the "input" function. Consequently, the application displays the identity entered earlier and the generation category, as shown in Picture 4(d). Furthermore, in the third meeting, students learned the use of looping functions ("for" and "while") and applied them in a mathematical context, as presented in Picture 5.



```
#Iterasi pada List
buah = ["Apel", "Mangga", "Jeruk", "Pisang", "Durian"]
for b in buah:
    print(f"Saya suka {b}")
```

```
Saya suka Apel
Saya suka Mangga
Saya suka Jeruk
Saya suka Pisang
Saya suka Durian
```

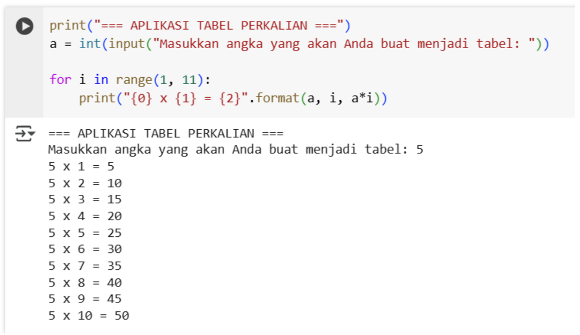
(a) Artifact examples on T_{3-1}



```
#Perulangan While
angka = 1
while angka <= 10:
    print(f"Data ke-{angka}")
    if angka == 3:
        print("Berhenti di angka 3")
        break
    angka += 1
```

```
Data ke-1
Data ke-2
Data ke-3
Berhenti di angka 3
```

(b) Artifact examples on T_{3-2}

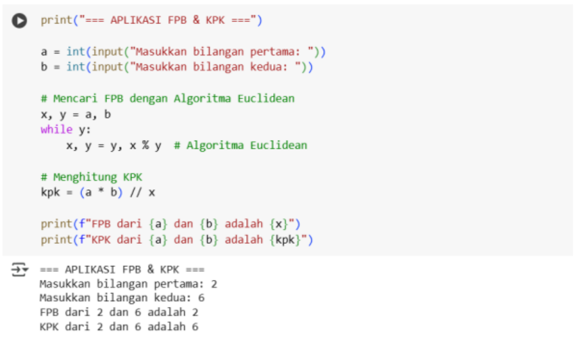


```
print("=== APLIKASI TABEL PERKALIAN ===")
a = int(input("Masukkan angka yang akan Anda buat menjadi tabel: "))

for i in range(1, 11):
    print("{0} x {1} = {2}".format(a, i, a*i))
```

```
=== APLIKASI TABEL PERKALIAN ===
Masukkan angka yang akan Anda buat menjadi tabel: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

(c) Artifact examples on T_{3-3}



```
print("=== APLIKASI FPB & KPK ===")

a = int(input("Masukkan bilangan pertama: "))
b = int(input("Masukkan bilangan kedua: "))

# Mencari FPB dengan Algoritma Euclidean
x, y = a, b
while y:
    x, y = y, x % y  # Algoritma Euclidean

# Menghitung KPK
kpk = (a * b) // x

print(f"FPB dari {a} dan {b} adalah {x}")
print(f"KPK dari {a} dan {b} adalah {kpk}")
```

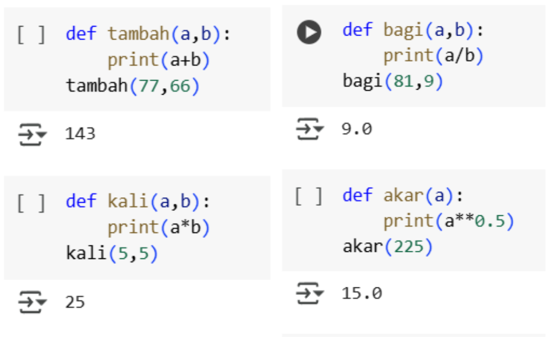
```
=== APLIKASI FPB & KPK ===
Masukkan bilangan pertama: 2
Masukkan bilangan kedua: 6
FPB dari 2 dan 6 adalah 2
KPK dari 2 dan 6 adalah 6
```

(d) Artifact examples on T_{3-4}

Picture 5. Coding artifact example on Task 3

Picture 5 illustrates the use of looping functions to support mathematical learning. The lesson begins by giving students the opportunity to try using the "for" ($T_{3\text{-}1}$) and "while" functions ($T_{3\text{-}2}$). Next, students were challenged to create a simple multiplication application ($T_{3\text{-}3}$), where the user could enter any number using the "input" function. As a result, the application displays the multiplication table of numbers from 1 to 10, as shown in Picture 5(c). In the final session ($T_{3\text{-}4}$), the students were challenged to create a simple application to find the least common multiple (LCM) and the great common divisor (GCD). The user is asked to input two numbers; the application displays the LCM and GCD, as presented in Picture 5(d).

The results of observations and interviews showed that some students still experienced "errors". After the examination, errors were caused by inaccuracies in writing the syntax. For example, after writing "for" or "while" at the end of the line, they did not add ":", so the program could not be run. Furthermore, in the last meeting of this study, students learned the use of the "def" function and applied it in supporting math learning, as presented in Picture 6.



```
[ ] def tambah(a,b):
      print(a+b)
    tambah(77,66)

143
```

```
def bagi(a,b):
  print(a/b)
bagi(81,9)

9.0
```

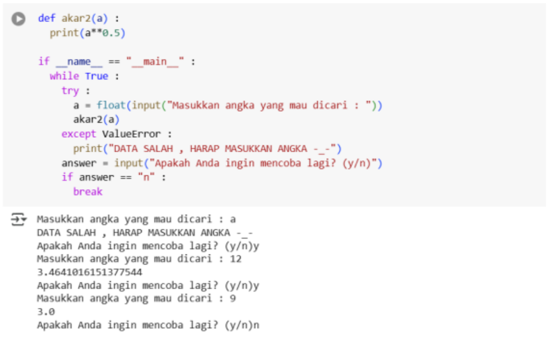
```
[ ] def kali(a,b):
      print(a*b)
    kali(5,5)

25
```

```
[ ] def akar(a):
      print(a**0.5)
    akar(225)

15.0
```

(a) Artifact examples on $T_{4\text{-}1}$

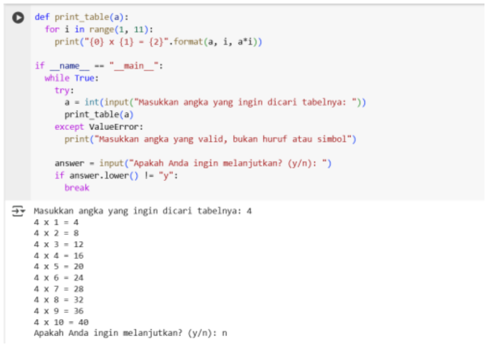


```
def akar2(a) :
  print(a**0.5)

if __name__ == "__main__" :
  while True :
    try :
      a = float(input("Masukkan angka yang mau dicari : "))
      akar2(a)
    except ValueError :
      print("DATA SALAH , HARAP MASUKKAN ANGKA -_-")
    answer = input("Apakah Anda ingin mencoba lagi? (y/n)")
    if answer == "n" :
      break
```

```
Masukkan angka yang mau dicari : a
DATA SALAH , HARAP MASUKKAN ANGKA -_-
Apakah Anda ingin mencoba lagi? (y/n)y
Masukkan angka yang mau dicari : 12
3.4641016151377544
Apakah Anda ingin mencoba lagi? (y/n)y
Masukkan angka yang mau dicari : 9
3.0
Apakah Anda ingin mencoba lagi? (y/n)n
```

(b) Artifact examples on $T_{4\text{-}2}$



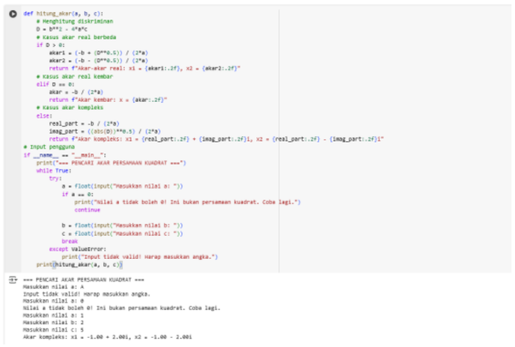
```
def print_table(a):
  for i in range(1, 11):
    print("{0} x {1} = {2}".format(a, i, a*i))

if __name__ == "__main__":
  while True:
    try:
      a = int(input("Masukkan angka yang ingin dicari tabelnya: "))
      print_table(a)
    except ValueError:
      print("Masukkan angka yang valid, bukan huruf atau simbol")

    answer = input("Apakah Anda ingin melanjutkan? (y/n): ")
    if answer.lower() != "y":
      break
```

```
Masukkan angka yang ingin dicari tabelnya: 4
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
Apakah Anda ingin melanjutkan? (y/n): n
```

(c) Artifact examples on $T_{4\text{-}3}$



```
def hitung_akar(a, b, c):
    # Menghitung diskriminan
    D = b**2 - 4*a*c
    # Kasus akar real berbeda
    if D > 0:
        akar1 = (-b + (D**0.5)) / (2*a)
        akar2 = (-b - (D**0.5)) / (2*a)
        return f"Akar-akar real: x1 = {akar1:.2f}, x2 = {akar2:.2f}"
    # Kasus akar real kembar
    elif D == 0:
        akar = -b / (2*a)
        return f"Akar kembar: x = {akar:.2f}"
    # Kasus akar kompleks
    else:
        real_part = -b / (2*a)
        imag_part = ((abs(D))**0.5) / (2*a)
        return f"Akar kompleks: x1 = {real_part:.2f} + {imag_part:.2f}i, x2 = {real_part:.2f} - {imag_part:.2f}i"
# Input pengguna
if __name__ == "__main__":
    print("=== PENCARI AKAR PERSAMAAN KUADRAT ===")
    while True:
        try:
            a = float(input("Masukkan nilai a: "))
            if a == 0:
                print("Nilai a tidak boleh 0! Ini bukan persamaan kuadrat. Coba lagi.")
                continue

            b = float(input("Masukkan nilai b: "))
            c = float(input("Masukkan nilai c: "))
            break
        except ValueError:
            print("Input tidak valid! Harap masukkan angka.")
    print(hitung_akar(a, b, c))
```

```
=== PENCARI AKAR PERSAMAAN KUADRAT ===
Masukkan nilai a: A
Input tidak valid! Harap masukkan angka.
Masukkan nilai a: 0
Nilai a tidak boleh 0! Ini bukan persamaan kuadrat. Coba lagi.
Masukkan nilai a: 1
Masukkan nilai b: 2
Masukkan nilai c: 5
Akar kompleks: x1 = -1.00 + 2.00i, x2 = -1.00 - 2.00i
```

(d) Artifact examples on $T_{4\text{-}4}$

Picture 6. Coding artifact example on Task 4

Picture 6 shows evidence of the utilization of the "def" function in supporting mathematical computation. The lesson started by giving students the opportunity to try using the "def" function ($T_{4\text{-}1}$). Next, students were challenged to create a simple application to find the root of a number ($T_{4\text{-}2}$). Students can create an application to find the root of a number, even by adding a looping function, where the program will keep running if the user does not end using the "break" function, as presented in Picture 6(b).

The developed application is slightly better than the previous one. If the user enters an incorrect input, such as a letter, a warning notification appears, which is made by utilising the “try” and “except ValueError” functions. Next, students are asked to create a simple multiplication application using the “def” function (T_{4-3}). The user can enter any number using the “input” function, and the application will display the multiplication table of that number from 1 to 10, as presented in Picture 6(c). In the last session, students were challenged to create a simple application to find the roots of a quadratic equation (T_{4-4}). As a result, students were able to create an application to determine the roots of a quadratic equation, as presented in Figure 6(d), by combining various functions previously learned.

Observations during the lesson found that some students experienced Python syntax “errors”. Upon inspection, the errors were caused by inaccuracies in the syntax writing. For example, after writing “if”, “elif”, and “else”, at the end of the line, they did not add a “:” sign, so the program could not be run. In addition, it was also found that some students did not pay attention to indentation while writing the program; for example, in the condition of writing “if”, “try”, or “while”, they did not indent the syntax writing, resulting in “errors”.

Overall, the results of the HLT implementation show that students can gradually understand the basic concepts of Python programming and apply them in mathematical contexts. The application of Google Colab as a learning medium provides advantages, especially in facilitating students who do not have personal devices (Ferreira et al., 2024). In addition, the automatic saving feature, ease of code-sharing, and cloud-based execution allow students to be more active in discussion and collaboration (Naik et al., 2021).

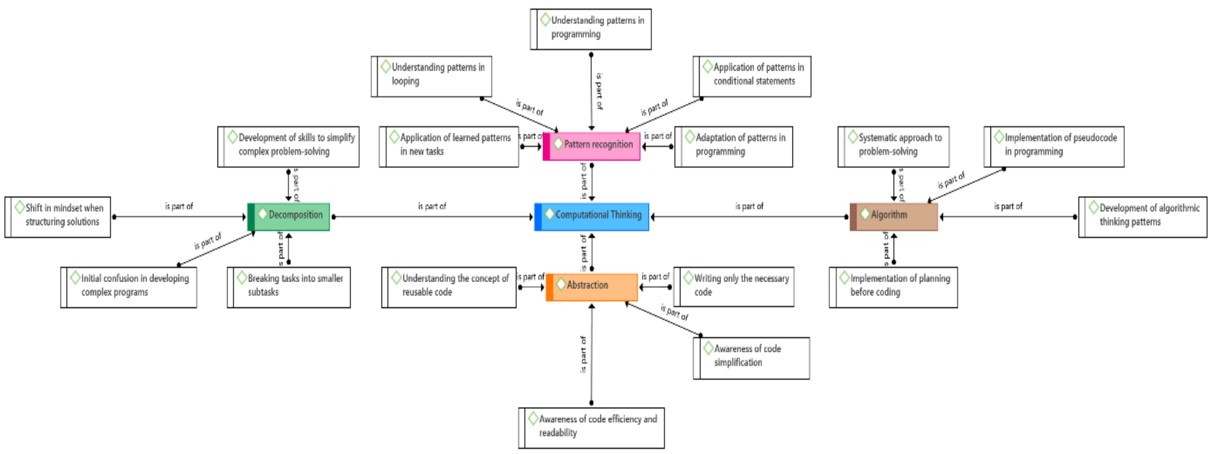
The analysis also showed that most students’ errors in writing code were not caused by an understanding of syntax or programming concepts but by a lack of accuracy in writing code. The most common errors were negligence in writing punctuation, such as lacking colons (:), quotation marks (" "), or incomplete parentheses () in function calls. In addition, errors in indentation often occur because students ignore the correct structure of code blocks according to Python rules. This finding aligns with the research of Liu et al. (2023), which showed that most errors in beginner programming are minor syntax errors due to a lack of attention to detail rather than difficulties in understanding algorithmic concepts.

Several efforts have been made to overcome some of these problems. First, live coding with error exploration was used. Lecturers write code to live by deliberately inserting common errors and then discussing them with students to train their error identification and correction skills (Strickroth, 2024). The second strategy is reflection-based debugging. Students are encouraged to analyse error messages that appear when running codes and record the errors they often make as a form of learning reflection (Fitzgerald et al., 2008; Yang et al., 2024). Third, peer code review. Students are asked to review their friends’ code, so they are more thorough in evaluating it and understanding the importance of syntax consistency and indentation (Lin et al., 2021). Finally, using the " explains the error" menu on Google Colab. Encouraging students to use tools such as "explain error" in order to detect and correct syntax errors and correct them easily because Gemini’s artificial intelligence facilities support it (Llerena-Izquierdo et al., 2024).

Retrospective Analysis

The third stage of this research is a more retrospective analysis, which aims to evaluate whether the HLT that has been developed and implemented previously can facilitate the achievement of learning objectives. In accordance with what has been determined previously, the development of this HLT aims to facilitate students in creating simple mathematical applications by utilising various basic functions in Python while honing their CT skills. The first objective of this HLT development has been proven to be achieved, where students have successfully created simple applications using Python, such as area and perimeter finding applications ($T_{1\text{-}4}$), value conversion applications ($T_{2\text{-}3}$), generation identification applications ($T_{2\text{-}4}$), multiplication table generation application ($T_{3\text{-}3}$), LCM and GCD finding applications ($T_{3\text{-}4}$) and ($T_{4\text{-}3}$), square root finding application of a number ($T_{4\text{-}2}$), and root finding application of a quadratic equation ($T_{4\text{-}4}$).

The identification of the role of programming utilisation in honing students' CT is based on coding artefacts and is supported by interview results. Both sets of data were analysed using Atlas.ti software and the findings are presented in Picture 7.


Understanding patterns in programming
Understanding patterns in looping
Application of patterns in conditional statements
Development of skills to simplify complex problem-solving
Application of learned patterns in new tasks
Pattern recognition
Adaptation of patterns in programming
Systematic approach to problem-solving
Implementation of pseudocode in programming
Shift in mindset when structuring solutions
Decomposition
Computational Thinking
Algorithm
Development of algorithmic thinking patterns
Initial confusion in developing complex programs
Breaking tasks into smaller subtasks
Understanding the concept of reusable code
Abstraction
Writing only the necessary code
Implementation of planning before coding
Awareness of code simplification
Awareness of code efficiency and readability
is part of

Picture 7. Concept map of CT development in a Python programming course

The research findings presented in Picture 7 show that, based on coding artifacts, observations, and interviews with students, it is known that learning programming can hone their CT skills. First, the initial confusion in developing complex programs reflects the indication that decomposition skills are honed out. However, they experience a change in mindset when developing solutions by dividing tasks into smaller parts, thus honing their skills in simplifying complex problems into simpler ones. Second, from the abstraction aspect, students' awareness of the importance of simplifying code, understanding the concept of reusable code, realising the importance of code efficiency and readability, and writing code that is really needed to solve the problem. Third, the development of pattern recognition is reflected in the understanding of how patterns are used in programming, using patterns in control structures such as if-else, recognising patterns in the use of loops (for, while), applying learned patterns to new tasks, and adjusting learned patterns to new programming contexts. Finally, students' algorithm skills were honed through efforts to plan before writing code, write pseudocode as the initial programming stage, build systematic, algorithmic thinking patterns, and use a systematic approach to solve problems. This finding shows that, in general, Python coding learning implemented in accordance with the HLT developed can hone students' CT skills.

From a CT perspective, this learning includes four main elements, as defined by Wing (2006, 2011) and further developed in recent studies (Shute et al., 2017; Weintrop et al., 2015). First, in terms of decomposition. Students learn to break down problems into small steps, such as developing programs to solve basic mathematical operations (Dong et al., 2019). Second, from the abstraction perspective. Students can simplify calculations and build modular programs (Grover & Pea, 2018) using functions and data structures. Third, we considered pattern recognition. By looping through multiplication tables and GCD, students identify patterns that can be reused in various programming scenarios (Tang et al., 2020). Finally, we consider the aspect of the algorithm. Students devise logical steps to reach a solution, such as devising algorithms for finding prime numbers or sorting data (Román-González et al., 2019).

This study's results reinforce previous studies' findings, which showed that the development of HLT can effectively facilitate the achievement of learning objectives (Andrews-Larson et al., 2017; Antonides & Battista, 2022; Clements et al., 2020; Gane et al., 2021; Irawan, 2024; Kuswardi et al., 2024; Rianasari & Guzon, 2024; Rich et al., 2022). In the future, the integration of more complex project-based learning can improve students' CT skills and deepen their understanding of Python programming in the context of mathematics learning (Bai et al., 2021; Rais & Xuezhi, 2024; Ye et al., 2023). In addition, implementing explicit debugging techniques as part of a learning strategy can help students improve their rigor when writing code (Fitzgerald et al., 2008; Yang et al., 2024).

Conclusions and Suggestions

This study seeks to develop HLT in a programming course using Python at Google Colab to hone prospective mathematics teachers' CT skills. The findings show that the developed HLT can facilitate the development of students' CT in terms of decomposition, abstraction, pattern recognition, and algorithms. Students showed an increased ability to solve complex problems in smaller parts, making them easier to solve. Students also improved in identifying important information needed to solve problems, patterns of solving a problem based on their experience, and the ability to design problem-solving algorithms systematically. In addition, the use of Python in Google Colab allows programming learning to be accessible to students who do not have laptops or computers. Overall, this study confirms that well-designed programming learning through the preparation of HLT can serve as an effective strategy to hone the CT skills of prospective mathematics teachers.

Despite its contributions, this study has some limitations that should be considered. The limited duration of teaching may not be sufficient to ensure the long-term sustainability and development of students' CT skills. In addition, this study was conducted specifically on prospective mathematics teacher students at one institution; therefore, the generalizability of the findings is limited to populations with similar characteristics. In addition, the assessment of the development of students' CT skills was analyzed qualitatively; therefore, generalization of the findings to a broader educational context was not possible. To overcome the shortcomings of this study, future research should combine quantitative methodology with standardized assessment instruments to provide a more comprehensive evaluation. In addition, further studies should examine the long-term impact of using HLT in teaching programming and explore the development of more interactive learning strategies to improve prospective mathematics teachers' CT skills.

Acknowledgements

All authors would like to thank the students of the Computer Programming course for their active participation and seriousness in learning Python using the HLT developed in this study.

References

Aho, A. V. (2012). Computation and Computational Thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Andrews-Larson, C., Wawro, M., & Zandieh, M. (2017). A Hypothetical Learning Trajectory for Conceptualizing Matrices as Linear Transformations. *International Journal of Mathematical Education in Science and Technology*, *48*(6), 809–829. https://doi.org/10.1080/0020739X.2016.1276225

Angeli, C., & Giannakos, M. (2020). Computational Thinking Education: Issues and Challenges. *Computers in Human Behavior*, *105*(106185), 1–8. https://doi.org/10.1016/j.chb.2019.106185

Antonides, J., & Battista, M. T. (2022). A Learning Trajectory for Enumerating Permutations: Applying and Elaborating a Theory of Levels of Abstraction. *The Journal of Mathematical Behavior*, *68*, 101010. https://doi.org/10.1016/j.jmathb.2022.101010

Badan Standar, Kurikulum, dan Asesmen Pendidikan Kementerian Pendidikan Dasar dan Menengah Republik Indonesia. (2025). *Naskah Akademik Pembelajaran Koding dan Kecerdasan Artifisial pada Pendidikan Dasar dan Menengah*. Pusat Kurikulum dan Pembelajaran & Pusat Standar dan Kebijakan Pendidikan Kemendikdasmen.

Bai, H., Wang, X., & Zhao, L. (2021). Effects of the Problem-Oriented Learning Model on Middle School Students' Computational Thinking Skills in a Python Course. *Frontiers in Psychology*, *12*(1), 1–14. https://doi.org/10.3389/fpsyg.2021.771221

Cansu, F. K., & Cansu, S. K. (2019). An Overview of Computational Thinking. *International Journal of Computer Science Education in Schools*, *3*(1), 17–30. https://doi.org/10.21585/ijcses.v3i1.53

Choi, W. C., & Choi, I. C. (2024). The Influence of Codecombat on Computational Thinking in Python Programming Learning at Primary School. *2024 5th International Conference on Education Development and Studies*, 26–32. https://doi.org/10.1145/3669947.3669951

Clements, D. H., & Sarama, J. (2024). Systematic Review of Learning Trajectories in Early Mathematics. *ZDM – Mathematics Education*, 14. https://doi.org/10.1007/s11858-024-01644-1

Clements, D. H., Sarama, J., Baroody, A. J., & Joswick, C. (2020). Efficacy of a Learning Trajectory Approach Compared to a Teach-to-Target Approach for Addition and Subtraction. *ZDM*, *52*(4), 637–648. https://doi.org/10.1007/s11858-019-01122-z

Corbin, J. M., & Strauss, A. L. (2015). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (Fourth edition). SAGE.

De Jesús, S., & Martinez, D. (2020). *Applied Computational Thinking with Python: Design Algorithmic Solutions for Complex and Challenging Real-World Problems*. Packt Publishing.

Denzin, N. K., & Lincoln, Y. S. (Eds.). (2018). *The SAGE Handbook of Qualitative Research* (Fifth edition). SAGE.

Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., Joshi, D., Robinson, R., & Andrews, A. (2019). PRADA: A Practical Model for Integrating Computational Thinking in K-12 Education. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 906–912. https://doi.org/10.1145/3287324.3287431

Downey, A. (2024). *Think Python: How to Think Like a Computer Scientist* (Third edition). O'Reilly Media, Inc.

Ferreira, R., Canesche, M., Jamieson, P., Neto, O. P. V., & Nacif, J. A. M. (2024). Examples and Tutorials on Using Google Colab and Gradio to Create Online Interactive Student-Learning Modules. *Computer Applications in Engineering Education*, *32*(4), e22729. https://doi.org/10.1002/cae.22729

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: Finding, Fixing and Flailing, a Multi-Institutional Study of Novice Debuggers. *Computer Science Education*, *18*(2), 93–116. https://doi.org/10.1080/08993400802114508

Gane, B. D., Israel, M., Elagha, N., Yan, W., Luo, F., & Pellegrino, J. W. (2021). Design and Validation of Learning Trajectory-Based Assessments for Computational Thinking in Upper Elementary Grades. *Computer Science Education*, *31*(2), 141–168. https://doi.org/10.1080/08993408.2021.1874221

Grover, S., & Pea, R. (2018). Computational Thinking: A Competency Whose Time Has Come. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Academic. https://doi.org/10.5040/9781350057142

Guillod, J. (2024). *Python Programming for Mathematics* (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781003565451

Hsiao, T.-C., Chuang, Y.-H., Chang, C.-Y., Chen, T.-L., Zhang, H.-B., & Chang, J.-C. (2023). Combining Building Block Process with Computational Thinking Improves Learning Outcomes of Python Programming with Peer Assessment. *Sage Open*, *13*(4). https://doi.org/10.1177/21582440231217715

Ilic, U., Haseski, H. I., & Tugtekin, U. (2018). Publication Trends Over 10 Years of Computational Thinking Research. *Contemporary Educational Technology*, *9*(2), 131–153. https://doi.org/10.30935/cet.414798

Irawan, E. (2024). *Keterampilan Computational Thinking Mahasiswa Melalui Penerapan Desain Didaktis Dengan Memanfaatkan Perangkat Lunak-R pada Mata Kuliah*

Statistika [PhD Thesis, Universitas Pendidikan Indonesia]. https://repository.upi.edu/

Irawan, E., & Herman, T. (2023). Trends in Research on Interconnection of Mathematics and Computational Thinking. *AIP Conference Proceedings*, *2805*, 1–9. https://doi.org/10.1063/5.0148018

Irawan, E., Rosjanuardi, R., & Prabawanto, S. (2024a). Promoting Computational Thinking Through Programming Trends, Tools, and Educational Approaches: A Systematic Review. *Jurnal Teori Dan Aplikasi Matematika*, *8*(4), 1327–1348. https://doi.org/10.31764/jtam.v8i4.26407

Irawan, E., Rosjanuardi, R., & Prabawanto, S. (2024b). Research Trends of Computational Thinking in Mathematics Learning: A Bibliometric Analysis from 2009 to 2023. *Eurasia Journal of Mathematics, Science and Technology Education*, *20*(3), 1–16. https://doi.org/10.29333/ejmste/14343

Jansen, P. (2025). *TIOBE Index*. Tiobe. https://www.tiobe.com/tiobe-index/

Jesús, S. de, & Martinez, D. (2023). *Applied Computational Thinking with Python: Algorithm Design for Complex Real-World Problems* (Second edition). Packt Publishing Ltd.

Kamak, L. P., & Mago, V. (2023). Assessing the Impact of Using Python to Teach Computational Thinking for Remote Schools in a Blended Learning Environment. In P. Zaphiris & A. Ioannou (Eds.), *Learning and Collaboration Technologies* (pp. 482–500). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-34550-0_35

Kim, J., Kim, M., Yu, H., Kim, Y., & Kim, J. (2019). Effect of Data Visualization Education with Using Python on Computational Thinking of Six Grade in Elementary School. *Journal of The Korean Association of Information Education*, *23*(3), 197–206.

Kong, S.-C., Lai, M., & Sun, D. (2020). Teacher Development in Computational Thinking: Design and Learning Outcomes of Programming Concepts, Practices and Pedagogy. *Computers & Education*, *151*, 103872. https://doi.org/10.1016/j.compedu.2020.103872

Kuroki, M. (2021). Using Python and Google Colab to Teach Undergraduate Microeconomic Theory. *International Review of Economics Education*, *38*, 100225. https://doi.org/10.1016/j.iree.2021.100225

Kuswardi, Y., Nurhasanah, F., Firdaus, N. U. A., Usodo, B., Chrisnawati, H. E., Sutopo, S., & Shahrill, M. (2024). A Learning Trajectory for Statistics Through the Traditional Game of Congklak to Enhance Mathematical Reasoning Skills. *International Journal of Pedagogy and Teacher Education*, *8*(1), Article 1. https://doi.org/10.20961/ijpte.v8i1.90547

Lin, X., Ma, Y., Ma, W., Liu, Y., & Tang, W. (2021). Using Peer Code Review to Improve Computational Thinking in a Blended Learning Environment: A Randomized Control Trial. *Computer Applications in Engineering Education*, *29*(6), 1825–1835. https://doi.org/10.1002/cae.22425

Liu, D., Feng, Y., Yan, Y., & Xu, B. (2023). Towards Understanding Bugs in Python Interpreters. *Empirical Software Engineering*, *28*(1), 19. https://doi.org/10.1007/s10664-022-10239-x

Llerena-Izquierdo, J., Mendez-Reyes, J., Ayala-Carabajo, R., & Andrade-Martinez, C. (2024). Innovations in Introductory Programming Education: The Role of AI with Google Colab and Gemini. *Education Sciences*, *14*(12), 1330. https://doi.org/10.3390/educsci14121330

Naik, P. G., Naik, G. R., & Patil, M. B. (2021). *Conceptualizing Python in Google Colab*. Shashwat Publication.

Palts, T., & Pedaste, M. (2020). A Model for Developing Computational Thinking Skills. *Informatics in Education*, *19*(1), 113–128. https://doi.org/10.15388/infedu.2020.06

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books.

Papert, S. (1996). An Exploration in the Space of Mathematics Educations. *International Journal of Computers for Mathematical Learning*, *1*(1). https://doi.org/10.1007/BF00191473

Rais, D., & Xuezhi, Z. (2024). Elevating Student Engagement and Academic Performance: A Quantitative Analysis of Python Programming Integration in the" Merdeka Belajar" Curriculum. *Journal on Mathematics Education*, *15*(2), 495–516.

Ren, H., Yang, L., Jiang, L., Bai, Y., Lu, W., & Chang, J. (2021). A Computational-Thinking-Oriented Progressive Teaching Mode for Python Course. *2021 IEEE 3rd International Conference on Computer Science and Educational Informatization (CSEI)*, 81–84. https://doi.org/10.1109/CSEI51395.2021.9477642

Rianasari, V. F., & Guzon, A. F. H. (2024). Designing Learning Trajectory to Support Preservice Mathematics Teachers' Skills in Creating and Implementing Realistic Mathematics Tasks. *Journal on Mathematics Education*, *15*(3), 701–716. https://doi.org/10.22342/jme.v15i3.pp701-716

Rich, K. M., Franklin, D., Strickland, C., Isaacs, A., & Eatinger, D. (2022). A Learning Trajectory for Variables Based in Computational Thinking Literature: Using Levels of Thinking to Develop Instruction. *Computer Science Education*, *32*(2), 213–234. https://doi.org/10.1080/08993408.2020.1866938

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017). K-8 Learning Trajectories Derived from Research Literature: Sequence, Repetition, Conditionals. *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 182–190. https://doi.org/10.1145/3105726.3106166

Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining Assessment Tools for a Comprehensive Evaluation of Computational Thinking Interventions. In S.-C. Kong & H. Abelson (Eds.), *Computational Thinking Education* (pp. 79–98). Springer Singapore. https://doi.org/10.1007/978-981-13-6528-7_6

Saha, A. (2015). *Doing Math with Python: Use Programming to Explore Algebra, Statistics, Calculus, and More!* No Starch Press.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Simon, M. A. (1995). Reconstructing Mathematics Pedagogy from a Constructivist Perspective. *Journal for Research in Mathematics Education*, *26*(2), 114–145. https://doi.org/10.5951/jresematheduc.26.2.0114

Simon, M. A. (2020). Hypothetical Learning Trajectories in Mathematics Education. In S. Lerman (Ed.), *Encyclopedia of Mathematics Education*. Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0

Simon, M. A., Kara, M., Placa, N., & Avitzur, A. (2018). Towards an Integrated Theory of Mathematics Conceptual Learning and Instructional Design: The Learning Through Activity Theoretical Framework. *The Journal of Mathematical Behavior*, *52*, 95–112. https://doi.org/10.1016/j.jmathb.2018.04.002

Simon, M. A., Placa, N., Kara, M., & Avitzur, A. (2018). Empirically-Based Hypothetical Learning Trajectories for Fraction Concepts: Products of the Learning Through Activity Research Program. *The Journal of Mathematical Behavior*, *52*, 188–200. https://doi.org/10.1016/j.jmathb.2018.03.003

Simon, M. A., & Tzur, R. (2004). Explicating the Role of Mathematical Tasks in Conceptual Learning: An Elaboration of the Hypothetical Learning Trajectory. *Mathematical Thinking and Learning*, *6*(2), 91–104. https://doi.org/10.1207/s15327833mtl0602_2

Strickroth, S. (2024). Scalable Feedback for Student Live Coding in Large Courses Using Automatic Error Grouping. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 499–505. https://doi.org/10.1145/3649217.3653620

Sun, L., & Zhou, L. (2023). Does Text-Based Programming Improve K-12 Students' Ct Skills? Evidence from a Meta-Analysis and Synthesis of Qualitative Data in Educational Contexts. *Thinking Skills and Creativity*, *49*, 101340.

Suryadi, D. (2013). Didactical Design Research (DDR) to Improve the Teaching of Mathematics. *Far East Journal of Mathematical Education*, *10*(1), 91–107.

Suryadi, D. (2019). *Penelitian Desain Didaktis (DDR) dan Implementasinya*. Gapura Press.

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing Computational Thinking: A Systematic Review of Empirical Studies. *Computers & Education*, *148*(1), 1–22. https://doi.org/10.1016/j.compedu.2019.103798

Tekdal, M. (2021). Trends and Development in Research on Computational Thinking. *Education and Information Technologies*, *26*(5), 6499–6529. https://doi.org/10.1007/s10639-021-10617-w

Vallejo, W., Díaz-Uribe, C., & Fajardo, C. (2022). Google Colab and Virtual Simulations: Practical E-Learning Tools to Support the Teaching of Thermodynamics and to Introduce Coding to Students. *ACS Omega*, *7*(8), 7421–7429. https://doi.org/10.1021/acsomega.2c00362

Wei, X., Lin, L., Meng, N., Tan, W., Kong, S.-C., & Kinshuk. (2021). The Effectiveness of Partial Pair Programming on Elementary School Students' Computational Thinking Skills and Self-Efficacy. *Computers & Education*, *160*(104023), 1–65. https://doi.org/10.1016/j.compedu.2020.104023

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2015). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2011). Research Notebook: Computational Thinking—What and Why? *Thelink*, 1–8.

Wing, J. M. (2017). Computational Thinking's Influence on Research and Education for All. *Italian Journal of Educational Technology*, *25*(2), 7–14. https://doi.org/10.17471/2499-4324/922

Yang, S., Baird, M., O'Rourke, E., Brennan, K., & Schneider, B. (2024). Decoding Debugging Instruction: A Systematic Literature Review of Debugging Interventions. *ACM Transactions on Computing Education*, *24*(4), 1–44. https://doi.org/10.1145/3690652

Ye, H., Liang, B., Ng, O.-L., & Chai, C. S. (2023). Integration of Computational Thinking in K-12 Mathematics Education: A Systematic Review on CT-Based Mathematics Instruction and Student Learning. *International Journal of STEM Education*, *10*(3), 1–26. https://doi.org/10.1186/s40594-023-00396-w

Zhuang, Y. Y., Kao, C.-W., & Yen, W.-H. (2025). A Static Analysis Approach for Detecting Array Shape Errors in Python. *Journal of Information Science & Engineering*, *41*(1). https://doi.org/10.6688/JISE.202501_41(1).0006