A Study of Value Trace Problems and Code Modification Problems in Python Programming Learning Assistant System

September, 2023

San Hay Mar Shwe

Graduate School of
Natural Science and Technology

(Doctor's Course)
Okayama University

Dissertation submitted to
Graduate School of Natural Science and Technology
of
Okayama University
for
partial fulfillment of the requirements
for the degree of
Doctor of Philosophy.

Written under the supervision of

Professor Nobuo Funabiki

and co-supervised by
Professor Satoshi Denno
and
Professor Yasuyuki Nogami

Okayama University, September 2023.

To Whom It May Concern

We hereby certify that this is a typical copy of the original doctor thesis of
Ms. San Hay Mar Shwe

Signature of	Seal of
the Supervisor	
	Graduate School of
Prof. Nobuo Funabiki	Natural Science and Technology

Abstract

Nowadays, *Python programming* has become very popular for various applications in both IT (information technology) and non-IT fields, due to rich libraries and short coding features. Then, low-hurdling learning tools have been highly demanded for self-study of introductory Python programming, since many users have no opportunities of taking Python programming courses at schools. Previously, we have studied *Java programming learning assistant system (JPLAS)* to assist self-study of Java programming.

Because of the importance of *Python programming* during these days, we are studying the *Python Programming Learning Assistant System (PyPLAS)* as a self-study tool for various *Python* programming topics. *PyPLAS* offers several types of exercise problems such as the *grammar-concept understanding problem (GUP)*, the *value trace problem (VTP)*, the *element fill-in-blank problem (EFP)*, and the *code writing problem (CWP)* to cover various students at different learning levels.

As the first contribution of the thesis, I implement the *value trace problem (VTP)* for novice learners to study grammar topics and basic programming skills through code reading. A *VTP* instance asks tracing the values of important variables or output messages in a given source code. The correctness of each answer is verified through *string matching* with the stored correct one. In addition, I make the references that describe the least *Python programming* topics related to the *VTP* instances, to assist novice learners in solving them.

However, any type of exercise problems in *PyPLAS* including *VTP* may not be suitable for data visualization studies such as graphs, because they need illustrating figures. Therefore, as the second contribution of the thesis, I introduce the *code modification problem (CMP)* for self-study of data visualization studies in *Python programming*, as a new problem type for *PyPLAS*. In the *CMP instance*, one source code and two images are given. The first image represents the output that is obtained by running the source code. Then, it asks to modify the source code to output the second image. The correctness of the answer is checked through string matching with the correct one.

Moreover, data analysis or data science is one of the important applications in Python programming. For data analysis, *Excel* is often used to manipulate the large amount of data within short durations and generate graphs as a powerful tool. It has become entrenched in business processes worldwide for diverse functions and applications. Therefore, as the third contribution of the thesis, I implement the *code modification problem (CMP)* for learning how to use Python programming libraries to manipulate data in Excel files. As a new function, a hint function is implemented for each *CMP* instance to assist learners in solving it.

In future works, I will continue studying other types of problems with useful functions for each problem type for other *Python programming* topics in *PyPLAS*.

Acknowledgements

It is my great pleasure to thank those who have supported and encouraged me this dissertation possible. Although it is very hard to express my gratitude with the proper words, I would like to convey my greatest blessing in my life.

Foremost, I owe my deepest gratitude to my supervisor, Professor Nobuo Funabiki, who has supported me throughout my thesis with his patience, motivation, encouragements, enthusiasm and meaningful suggestions. I am greatly indebted to him, whose countless valuable suggestions and advice from the beginning to the end enabled me to proceed this study, also even in daily life in Japan. Moreover, he always gave me very precious ideas with fruitful kindness and considerations about that how to survive in the study in Japan and how should be continued for future plans. Thanks for making me, I received a lot of motivations and energy to complete my research papers and for future survivals in life. Moreover, together with Mrs. Naomi Funabiki, he gave me suggestions for solving ways in my social life in Japan. This makes me to feel warm and safety environments, even I am far away from my family. Needless to say, it would not be possible to complete this thesis without his guidance and active support.

I am deeply grateful to my co-supervisors, Professor Satoshi Denno and Professor Yasuyuki Nogami, for their continuous supports, guidance, mindful suggestions, and proofreading of this work. I wish to express my sincere gratitude to Associate Professor Minoru Kuribayashi for his valuable suggestions during my research, and great ideas for composing wonderful presentations. I would like to express my gratitude to the course teachers during my Ph.D. study for enlightening me with wonderful knowledge. And, I would like to convey my appreciation to Ms. Keiko Kawabata for supporting required documents and necessary things during my study.

I would like to acknowledge the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT) for financially supporting my Ph.D. study, and all my respectful teachers in the University of Technology (Yadanarpon Cyber City) for guiding a lot of valuable knowledge.

I would like to thank my friends, especially Myanmar friends, and colleagues from Japanese class, and all the FUNABIKI Lab's members who helped me in this study and shared the time with me to have great experiences and unforgettable memories. I appreciate the supports at my tough time during this study and the thoughts and experiences shared with me. I would like to tell my special thanks to Dr. Htoo Htoo Sandi Kyaw, who gave me a lot of valuable advices and supports to start my Ph.D study. I would like to say my special thanks to Ms. Ei Ei Htet. She supported me a lot for everything like a sister.

Last but not least, I am eternally grateful to my beloved family and my soulmates, who are very important to my existence. Without them, it would be not easy to survive my life. They always encourage and support me not only in my study but also throughout my life. Your supports and understanding gave me the strength, inspirations and peacefulness to share my enjoy moments and overcome any difficulty in my life. I am blissful to have you all.

List of Publications

Journal Paper

1. **San Hay Mar Shwe**, Nobuo Funabiki, Yan Watequlis Syaifudin, Ei Ei Htet, Htoo Htoo Sandi Kyaw, Phyu Phyu Tar, Nandar Win Min, Thandar Myint, Hnin Aye Thant, and Wen-Chung Kao, “ Value trace problems with assisting references for Python programming self-study,” International Journal of Web Information Systems, June 2021.

International Conference Paper

2. **San Hay Mar Shwe**, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, Khaing Hsu Wai and Wen-Chung Kao, “ A proposal of code modification problem for Python programming learning assistant system,” International Symposium on Socially and Technically Symbiotic Systems (STSS), November 15-17, 2021.

3. **San Hay Mar Shwe**, Nobuo Funabiki, Khaing Hsu Wai, Shune Lae Aung, and Wen-Chung Kao, “A study of code modification problems for Excel operations in Python programming learning assistant system,” 2022 10th International Conference on Information and Education Technology (ICIET 2022), pp. 209-213, April 9-11, 2022.

List of Figures

List of Tables

List of Codes

Contents

Chapter 1

Introduction

1.1 Background

Information Technology (IT) has become increasingly important in recent years due to its pervasive presence in our lives and its role in driving innovation and efficiency. *IT* encompasses a broad range of activities related to managements, processing, and disseminations of various information using computer systems and networks. It plays a crucial role in a lot of sectors, including business, education, healthcare, communication, and entertainment. The rapid advancements in *IT* have transformed the way we live, work, and interact with the world around us, making *IT* skills increasingly important.

Teaching and learning programming languages becomes a crucial component of *IT* educations and professional developments for individuals who want to work in *IT* fields, as it enables them to develop the technical skills needed to create and maintain software systems. Recently, low-cost, low-hurdling learning tools have been highly demanded for self-studies of programming languages due to high demands for learning and teaching programming in the classroom lectures. A teacher may be overloaded with the several student's result verification and feedback comments during the short period. A student may have no opportunities of taking programming courses at schools.

We have developed the *Java programming learning assistant system (JPLAS)* as a self-study tool for learning *Java* programming. Both the online version and the desktop-based offline version have been implemented for *JPLAS* to deal with various environments. The online *JPLAS* adopts Ubuntu for the operating system, Tomcat for the Web application server, JSP for the application programs with HTML, and MySQL for the database for handling the problem descriptions and the students' data. The user can access to *JPLAS* through a web browser. Since the online *JPLAS* may not be suitable for students in poor Internet access environments, the offline *JPLAS* called the *Desktop-version JPLAS (D-JPLAS)* has been implemented without the online database and the web server [1]. In *D-JPLAS*, the assignment files and the answer text files can be shared between the teacher and the students using USB memories, a file server or email.

In *JPLAS*, there are several types of exercise problems, such as the *grammar concept understanding problem (GUP)* [2], for studying keyword definitions, the *value trace problem (VTP)* [3], for studying how to use keywords in the source code, the *element fill-in-blank problem (EFP)* [4], with studying partially writing source code, and the *code writing problem (CWP)*, for studying fully writing source code to cover various students at different learning levels. The correctness of any answer is automatically checked in the system, and the result will be automatically returned to the learner so that he/she can instantly find the mistakes and correct them.

Recently, *Python programming* has gained the significant popularity and importance for use in various groundbreaking fields in experiments, prototyping, embedded systems, and data sciences, due to rich libraries and short coding features. The simplicity, versatility, strong community supports, and relevances in emerging fields like data science and machine learning in *Python programming* have contributed to its popularity and importance in the current technology landscape. Therefore, we extended our previous *JPLAS* as *Python Programming Learning Assistant System (PyPLAS)* for learning *Python programming*.

1.2 Contributions

Motivated by the above background in *Python programming*, in this thesis, I propose three improvements in *Python Programming Learning Assistant System (PyPLAS)* as contributions of this thesis.

As the first contribution of the thesis, I present the *value trace problem (VTP)* for novice learners to study grammar topics and basic programming skills through code reading. A *VTP* instance asks a learner to trace the actual values of important variables or output messages in the given source code with the design goals: 1) a variety of useful source codes for *Python programming* study is depicted with full forms to novice learners, 2) a set of references describing the least *Python programming* topics related to the *VTP* instances are provided to assist novice learners, 3) learners can correctly answer the questions by carefully reading and understanding the source codes, and 4) any answer can be marked through string matching automatically.

For evaluations, 130 *VTP* instances are generated using *Python codes* in textbooks [24] and websites [5] that cover basic/advanced grammar topics, fundamental data structures and algorithms, and two common library usages (pandas and numpy libraries). Besides, assisting references on *Python programming* topics related to the *VTP* instances are introduced to assist novice learners in solving them efficiently. The applications to 48 undergraduate students in *Myanmar and Indonesia* confirm the validity of the proposal in *Python programming* self-studies by novice learners.

However, any type of exercise problems in *PyPLAS* including *VTP* may not suitable for *data visualization* study using various graphs, because they need illustrating figures. Therefore, as the second contribution of the thesis, I present the *code modification problem (CMP)* for self-study of *Python programming*, as a new problem type for *PyPLAS*. In the *CMP* instance, one source code and two images are given. The first image represents the output that is obtained by running the source code. Then, it asks to modify the source code to output the second image. The correctness of the answer is checked through string matching with the correct one.

For evaluations, a total of 25 *CMP* instances are generated to cover most important visualization concepts in Python programming. First, I collect the relevant source codes from the website [6]and the text book [7]. Then, I analyze important parts of the source codes that should be modified for understanding of visualization techniques clearly. After that, I generate the corresponding *CMP* instances with *HTML/CSS/JavaScript files* for the offline answering function of *PyPLAS*. To evaluate the generated 25 *CMP* instances, I assign them to 22 students in *Okayama University, Japan*. Their solution results show that the average correct rate for all of the 25 instances is at least 97%. Thus, the difficulty levels of the *CMP* instances are proper for the novice learners as the viable self-study learning tool.

Moreover, data science and data analysis are important fields in today's world due to the increasing availability of data, the need for business to remain competitive, the importance of data in scientific research and so on. Therefore, data analysis or data science is one of the important applications in *Python programming*. For data analysis, *Excel* is often used to manipulate the large amount of data within short duration and generate graphs as a powerful tool. Using *Python* with *Excel* operations in data analysis and data science enhances the capabilities of *Excel* by enabling to leverage *Python's* extensive libraries for data manipulation, analysis, modeling, visualization, and automation.

Therefore, as the third contribution of the thesis, we implemented *CMP* for learning how to use *Python programming* libraries to manipulate data in *Excel* files. As a new function, a hint function is also implemented for each *CMP* instance to assist learners in solving it. The goal of this study is to give the students about the opportunities of learning how to use *Python programming libraries* to manipulate data in *Excel* files with: 1) a variety of useful and practical source codes for studying *Python Excel operations* is depicted with full forms to novice learners, 2) a learner can correctly answer the questions of *CMP* by carefully reading the source code and modifying the required parts in functions, variables, and parameters 3) any answer can be marked through string matching automatically.

For evaluations, I generated 25 *CMP* instances using *Python* codes for various *Excel* operations using *pandas* and confirmed the validity from the application results to students in *Okayama University*.

Moreover, to verify that whether or not each CMP instance is understood by the learner, I introduce the project assignments. After solving the CMP instances completely, the learners need to solve the project assignments for each topic, respectively. In future works, I will continue studying other types of problems with useful functions for each problem type for other Python programming topics in *PyPLAS*.

1.3 Contents of This Dissertation

The remaining part of the thesis is organized as follows: In Chapter 2, I review the overview of *Java Programming Learning Assistant System (JPLAS)* with the explanation of server platform, software architecture, implemented problem types. I also review various service functions, online and offline versions of JPLAS, and elaboration of *PyPLAS* in Chapter 2 . In Chapter 3 , I present the value trace problem for five topics of Python programming. In Chapter 4, I define the code modification problem, the generation procedure and two level answer marking function for CMP. In Chapter 5, I present the code modification problem for studying data visualization concepts and evaluate student answer results. In Chapter 6, I present the code modification problem for studying Excel operations with the hint function implementation and analyze their application results. In Chapter 7, I review the related works in literature. In Chapter 8, I conclude this thesis with some future works.

Chapter 2

Review of Programming Learning Assistant System

In this chapter, we introduce the outlines of *Java Programming Learning Assistant System (JPLAS)* and *Python Programming Learning Assistant System (PyPLAS)*.

2.1 Overview of JPLAS

Firstly, we overview the server platform, the software architecture, and the implemented problem types of *JPLAS*.

2.1.1 Server Platform

Originally, *JPLAS* was implemented using *JSP* with *Java* 1.6.2 as the web application on a server. It adopts *Ubuntu-Linux 10.04* as the operating system running on *VMware* for portability. *Tomcat 6.0.26* is used as the web application server to run *JSP* source codes that is a script language with embedding *Java* codes within *HTML* codes. *Tomcat* returns the dynamically generated web pages to the client web browser. *MySQL 5.0.27* is adopted for managing the data in *JPLAS*. Figure 2.1 illustrates the server platform of *JPLAS* [8].

2.1.2 Software Architecture

The software architecture of *JPLAS* follows the MVC model as the common architecture of the web application system. It basically uses *Java* for the Model (M), *HTML/CSS/JavaScript* for the View (V), and *JSP* for the controller (C). The system implements the logic functions of *JPLAS* using *Java*. For the independence from the view and controller, any input/output to/from the model uses a string or its array that does not contain *HTML* tags. *Servlet* is not used to avoid the possible redundancy that could happen between *Java* codes and *Servlet* codes where the same function may be implemented. A design pattern called *Responsibility Chain* is adopted to handle marking functions of the student answers, and the specific functions for the database access are implemented such that the controller does not handle them. The view implements the user interfaces of *JPLAS* by using a *CSS* framework to provide integrated interfaces using *Cascading Style Sheet (CSS)* in the web standard. The user interface is dynamically controlled with *Ajax* to reduce the number of *JSP* files. For the control architecture, the control in *JPLAS* is implemented by *JSP*. When it

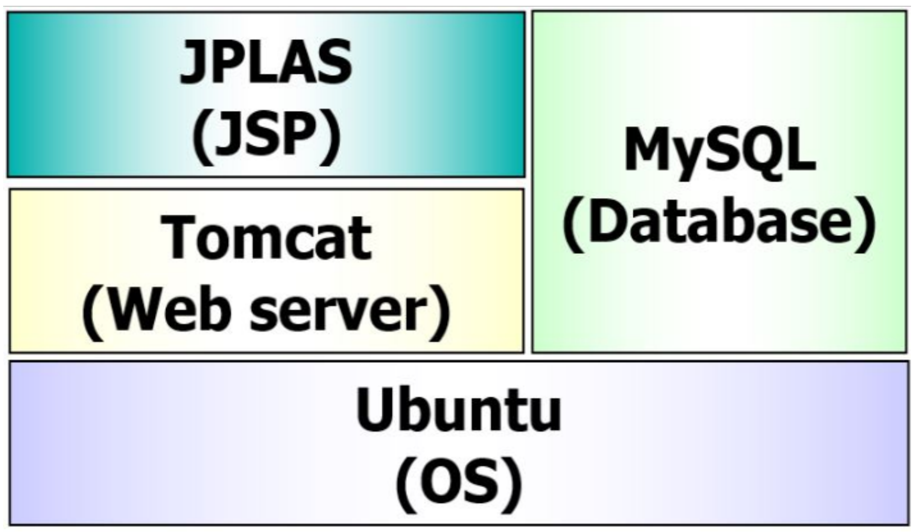


Figure 2.1: Server platform for JPLAS.

receives a request from the view, it sends it to *Java* in the model and requests the corresponding process. When *Java* in the model returns the processing results by strings, the control changes the format for the view use *HTML*. The procedure is elaborated as follows:

1) to show the assignment list in the view, *JSP* in the control receives the list with strings in the two dimensional array, changes them into the table format in *HTML*, and sends them to *JavaScript* in the view,

2) to demonstrate the selected assignment in the view, *JSP* receives the details with strings, changes them into the table in *HTML*, and sends it to *JavaScript*, and

3) to mark the answers from the student, *JSP* receives them from *JavaScript* in the view and sends them to *Java* in the model. After completing the marking in the model, *JSP* receives the marking results from *Java*, changes them into the table format in *HTML*, and sends it to *JavaScript* in the view [9].

The overall software architecture in *JPLAS* can be seen in Figure 2.2.

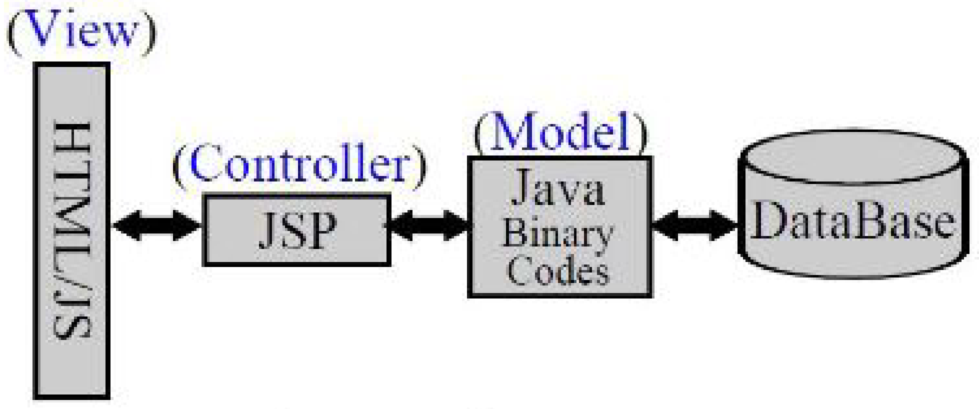


Figure 2.2: Software architecture for JPLAS.

2.1.3 Implemented Problem Types

Currently, *JPLAS* has several types of exercise problems to accommodate a variety of students at difference learning levels. Problem types in *JPLAS* are as follows:

1) *Grammar Concept Understanding Problem (GUP)*: This problem instance consists of a source code and a set of questions on grammar concepts or behaviors of the code. Each answer can be a number, a word, or a short sentence, whose correctness is marked through string matching with the correct one. Also, the algorithm is used to automatically generate a *GUP* instance from a given source code by: 1) extracting the registered keywords in the source code, 2) selecting the registered question corresponding to each extracted keyword, and 3) detecting the data required in the correct answer from the code [2].

2) *Value Trace Problem (VTP)*: This problem requires students to trace the actual values of important variables in a code when it is executed. The correctness of the answers is also marked by comparing them with their correct ones stored in the server [3].

3) *Element Fill-in-blank Problem (EFP)* : This problem requires students to fill in the blank elements in a given *Java* code. The correctness of the answers is marked by comparing them with their original elements in the code that are stored in the server. The original elements are expected to be the unique correct answers for the blanks [4]. To help a teacher generate a feasible element fill-in-blank problem, the blank element selection algorithm has been proposed [11].

4) *Statement Fill-in-blank Problem (SFB)*: This problem asks students to fill in the blank statements in a code. The correctness of the code is marked by using the test code on *JUnit* that is an open source software for the *test-driven development (TDD)* method [12]. To help a teacher select blank statements from a code, the *program dependency graph (PDG)* has been used [10].

5) *Code Writing Problem (CWP)*: This problem asks students to write a whole code from scratch that satisfies the specifications described in the test code [8]. The correctness of the code of students is also marked by the test code.

6) *Code Amendment Problem (CAP)*: In this problem type, a *Java* source code that has several missing or error elements, called a problem code, is shown to student. A student needs to identify the locations of missing or error elements in the code, and to fill in them or correct them with the correct elements. The correctness of any answer will be marked through string matching of the whole statement with the corresponding original one in the code [13].

7) *Code Completion Problem (CCP)*: In this problem, a source code with several missing elements is shown to the students without specifying their existences. Then, a student needs to locate the missing elements in the code and fill in the correct ones there. The correctness of the answer from a student is verified by applying string matching to each statement in the answer to the corresponding original statement in the code. Only if the whole statement is matched, the answer for the statement will become correct. Moreover, merely one incorrect element will result in the incorrect answer [14].

2.2 Service Functions in JPLAS

There are two services functions in *JPLAS*, namely, teacher service functions, and student service functions.

2.2.1 Teacher Service Functions

The teacher service functions include the problem generation, the registration and management of assignments, the creation of projects, and analyzing student performance by checking student's answers and viewing the number of submissions for individual problem by each student to evaluate the difficulty of assignments, and compressions of students. If most of the students did a lot of submissions for an assignment, the teacher need to consider that it will be too difficult for the learners and if it is necessary, the teacher needs to change or replace that problem with easier one. Sometimes, the teacher can implement hint functions, and recommendation functions for the assignments to assist the students for better understanding. In addition, the teacher can also create the references for each topic. On the other hands, if the teacher finds a student who submitted the answers many times whereas other students did so fewer times, in this case, the teacher needs to carefully instruct that student and check that student's performing extraordinarily. Moreover, the teacher can create the project assignment by summarizing all important concepts that was previously learned by the students to verify and evaluate the students' situations about the corresponding topics.

2.2.2 Student Service Functions

The student service functions include the view of the assignments, solving project assignments, and the submissions of their answers for the assignments. For code writing problem type, the student needs to write a source code for an assignment by reading problem statement, and the test code where the student must use the class/method names, the types, and the other specifications to satisfy the given test code. The answer from a student is generally processed at the *JPLAS* by the following steps:

1) When a student accesses to *JPLAS*, the list of the assigned problems to the student is displayed.
2) When a problem is selected by the student, the corresponding problem text in the database is displayed.
3) The student writes the answers in the corresponding forms.
4) The answers submitted by the student are marked in the server, and both the answer and the marking results will be saved in the database.
5) *JPLAS* offers feeds back to the student.
6) If necessary, the student could repeat the steps from (3).

The utilization procedure for both *JPLAS* functions by a teacher and a student are given as follows:

1) A teacher generates a new problem, and registers it to the database.

2) A teacher generates a new assignment by selecting proper problems in the database and registers it to the database.

3) A student selects an assignment to be solved.

4) A student selects a problem in the assignment to be solved.

5) A student solves the questions in the problem and submits the answers to the server.

6) The server marks the answers and returns the marking results.

7) A student modifies the incorrect answers and resubmits them to the server, if necessary.

8) A student refers to his/her solution results of the assignments.

9) A teacher refers to the solution results of all the students of the assignments.

2.3 Desktop-version JPLAS

As the previously mentions, *JPLAS* has been developed as a web application system. However, it has been found that the online system can be used only in Internet-available environments and it will be difficult in some areas where the Internet connection can't access at all or may not be stable due to the weak network infrastructure and the frequent power shortage, particularly in developing countries. To avoid those difficulties of the online *JPLAS*, we have implemented the offline *Desktop-version JPLAS (D-JPLAS)* as an efficient solution for schools and homes with the poor Internet accesses. Unlike to the *online JPLAS*, *D-JPLAS* runs on the client PC only, without the server access through the Internet. It keeps all the programs and data including the problems and the student answers in the file system of the user's PC, where it does not use the database.

Basically, the usage flow of offline *D-JPLAS* can be process through the following steps. Firstly, the teacher needs to create and assign the programming assignment. After that, he/she distributes the created assignment problems to the students, who are learning programming languages for improving their skills. Students can solve the assignments repeatedly on their own PC on offline until they can get the correct answers. After that, the students need to submit their answer files to the teacher who stores the files in the respective folder for each problem type and the student. Finally, the teacher will manage and analyze the submitted answers on his/her own PC using the answer analysis function, and give feedbacks to the students. The file exchange between the teacher and student will be done through the USB memories, the file servers, or emails if the Internet is accessible. Figure 2.3 illustrates the usage flow of *Desktop-version JPLAS* and Section 2.4 will discuss the offline answering functions in *JPLAS*.

2.4 Offline Answering Functions in JPLAS

As mentioned in Section 2.3, in addition to the online platform, the offline answering function has been implemented to allow students to answer the problems in *JPLAS* even if the students cannot access to the *JPLAS* server when the Internet is unavailable. Therefore, this function is very useful and actually inevitable in applying *JPLAS*. For solving the problem instances in this offline *JPLAS*, the problem assignment delivery and answer submission can be accomplished with a USB. There are three mainly functions: operation flow, file generation, and cheating prevention in *offline JPLAS*.

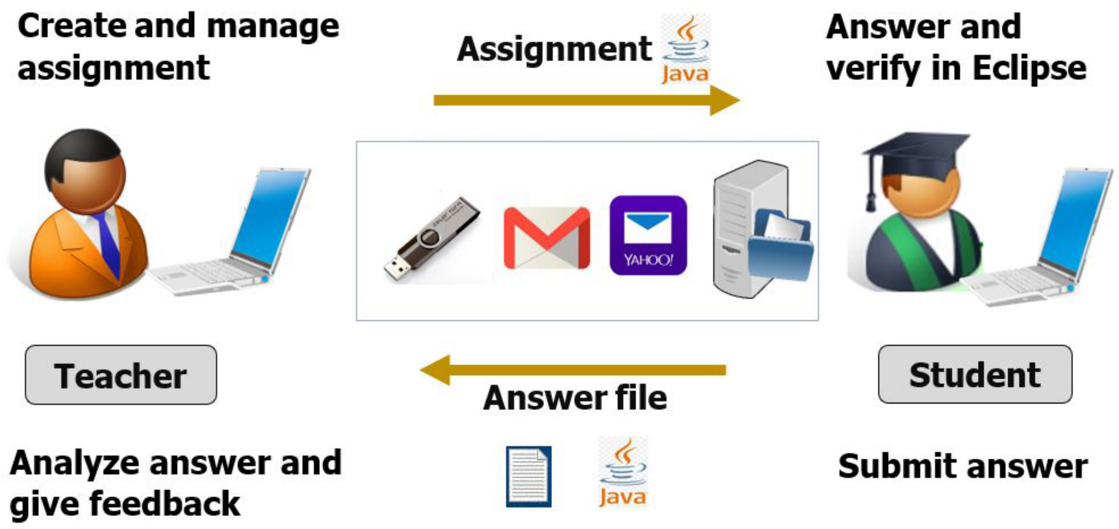

Create and manage assignment
Assignment
Java
Answer and verify in Eclipse
YAHOO!
Teacher
Answer file
Student
Analyze answer and give feedback
Java
Submit answer

Figure 2.3: Usage flow of Desktop-version JPLAS.

2.4.1 Operation Flow

The operation flow of the offline answering function is as follows:

1) Problem instance download: a teacher accesses to the *JPLAS* server, selects the problem instances for the assignment, and downloads the required files into the own PC on online.

2) Assignment distribution: the teacher distributes the assignment files to the students by using a file server or USB memories.

3) Assignment answering: the students receive and install the files on their PCs, and answer the problem instances in the assignment using Web browsers on offline, where the correctness of each answer is verified instantly at the browsers using the *JavaScript* program.

4) Answering result submission: the students submit their final answering results to the teacher by using a file server or USB memories.

5) Answering result upload: the teacher uploads the answering results from the students to the *JPLAS* server to manage them.

2.4.2 File Generation

Table 2.1 shows the necessary files with their specifications for the offline answering function in *JPLAS*. These files are designed for the problem view, the answer marking, and the answer storage.

2.4.3 Cheating Prevention

In *offline JPLAS*, the correct answers need to be distributed to the students so that their answers can be verified instantly on the browser. To prevent disclosing the correct answers, they will be distributed after taking hash values using *SHA256* [15]. In addition, to avoid generating the same hash values for the same correct answers, the assignment ID and the problem ID are concatenated with each correct answer before hashing. Then, the same correct answers for different blanks are converted to different hash values, which ensure the independence among blanks [14].

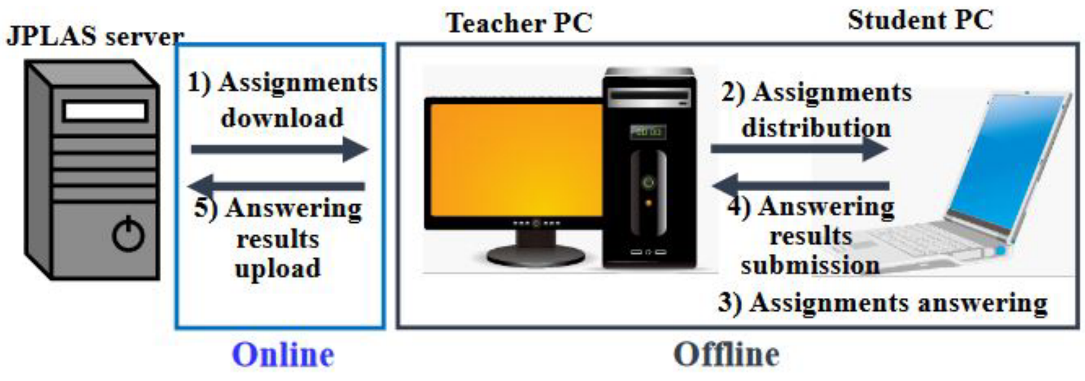


Figure 2.4: Operation flow for offline answering function.

Table 2.1: Files for distribution.

File name	Outline
css	CSS file for Web browser
index.html	HTML file for Web browser
page.html	HTML file for correct answers
jplas2015.js	js file for reading the problem list
distinction.js	js file for checking the correctness of answer
jquery.js	js file for use of jQuery
sha256	js file for use of SHA256
storage.js	js file for Web storage

2.5 Implementation of JPLAS Platform Using Node.js and Docker

Besides the *online JPLAS* and *offline JPLAS*, we implemented the *JPLAS* platform with the newly designed software architecture using *Node.js* and *Docker* without the internet connection, to avoid the redundancy and improve the portability of previous implementations [16].

The students can solve the *JPLAS* problems without Internet connection by installing all the system in their PCs. *Node.js* [17] is adopted as the popular web application server, where application programs on both the server and client can be made using *JavaScript*. Besides, *Express.js* [18] is used together as the framework to reduce the implementation cost of this platform. Furthermore, the user interface is dynamically controlled with *EJS* that can avoid the complex syntax structure.

To avoid the software version problem on a PC when we distribute the system to the students, we use *Docker* which provides the flexibility and portability for running various software in different platforms. *Docker* [19]is adopted to make students easily install the platform software in their own PCs, so that they can solve exercises in *JPLAS* without the Internet connections. *Docker* has been designed to make it easier to create, deploy, and run an application program on various platforms using the container. The *Docker* container [20] allows an application developer to combine all the necessary software required to run the application program, such as the libraries, the middleware, the parameters, and the other dependencies, into one package file called the container image, to be shipped out. The *Docker container image* is a lightweight, standalone, and executable package of all the software needed to run the application program. It may include the source codes, the runtime environments, the system tools, the system libraries, and the settings.

As the software architecture, *Mac OS* is adopted for the operating system in the server platform. *Node.js* is used as a web application server together with the *Express.js* framework. *EJS* is used for the template engine. Any database system is not installed for managing the data. JUnit [21] is used for testing the answer source codes in code writing problem [8]. *Visual Studio Code (VS Code)* IDE is used for editing the source codes as a popular development environment. In our architecture, *Java* is used for the model (M) to run *JUnit*, *JS/CSS/JavaScript* are for the view (V), and *JavaScript (Node/Express)* is for the controller(C). It is a compact web application server and can create both the client and server side of the application using only *JavaScript*. It can make application program is easy, simple and reduce by using a flexible framework as *Express.js* that provides ready-made components for a web application. *Node.js*. The overall architecture can be seen in Figure 2.5.

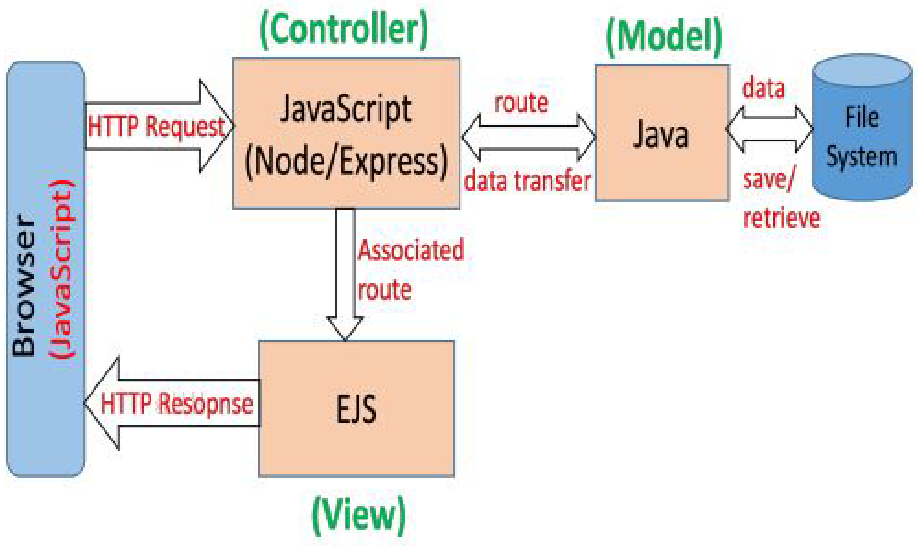


Figure 2.5: MVC model in JPLAS using Node.js.

2.6 Elaboration of PyPLAS

In today's technology landscape, *Python* becomes a widely used and highly versatile programming language that has gained immense popularity and importance in various domains. Some key reasons why *Python* is considered an important programming language can be seen in the following facts:

1. Easy to Learn and Readability

2. Wide Range of Applications

3. Large Standard Library and Third-Party Packages
4. Cross-Platform Compatibility
5. Strong Community and Support
6. Data Science and Machine Learning
7. Scripting and Automation
8. Integration and Extensibility

Therefore, *Python* is used by a great number of professionals, not just programmers or developers. The following lists are just a few of the careers where *Python* is a key skill:

- Back-end developer (server-side)
- Front-end developer (client-side)
- Full-stack developer (both client and server-side)
- Web designer
- Back-end developer (*Python* developer)
- Machine learning engineer
- Data scientist
- Data analyst
- Data engineer
- DevOps engineer (development operations)
- Software engineer
- Game developer
- Statistician
- SEO specialist
- And more… [22]

Due to the *Python's* importance, we considered to extend our *JPLAS* to *Python programming* learning, called *Python Programming Learning Assistant System (PyPLAS).*

2.6.1 Problem Types in PyPLAS

Currently, we have implemented various types of problems with automatic marking functions to cover self-studies of *Python programming* at different levels by novice students. Namely, they are *Grammar Concept Understanding (GUP), Value Trace Problem (VTP), Comment Insertion Problem (CIP), Code Modification Problem (CMP), and Code Writing Problem (CWP).* Among them, this thesis focuses on the VTP and CWP. In most of the problem types, the answers of students are marked by string matching with correct codes and in some problem types, the answers are marked by unit testing using test codes.

2.7 Summary

In this chapter, we reviewed *JPLAS*, including the functions in *JPLAS* and desktop versions of *JPLAS*. Also, this chapter presented implementation of the *JPLAS* platform using *Node.js* and *Docker*, and the elaboration of *PyPLAS* as the extension of *JPLAS*.

Chapter 3

Value Trace Problem

In this chapter, we present the *Value Trace Problem (VTP)* in *Python Programming Learning Assistant System (PyPLAS)* [23].

3.1 Introduction

The *value trace problem (VTP)* for self-study of *Python programming* asks a learner to trace the actual values of important variables or output message in the given source code. Then, the correctness of each answer is verified through string matching with the stored correct one. In this study, we generated a total of 130 *VTP* instances to cover grammar concepts, fundamental data structures or algorithms, and common libraries in *Python* programming. We collect the relevant source codes from the websites [5] and a text book [24]. We also make the references that describe the least *Python programming* topics related to the *VTP* instances, to assist novice learners in solving them. To evaluate the generated *VTP* instances, we assign them to 48 undergraduate students in Myanmar, and Indonesia, most of them have never studied *Python programming* in the past. This chapter will be organized as follows: the design of the *VTP* is first introduced. Then, the generations of *VTP* instances and their evaluations through applications to students are presented respectively. Finally, the conclusion with future work is given for this chapter.

3.2 Design of Value Trace Problem for Python Programming

In this section, we present the design of the *the value trace problem (VTP)* for *Python programming*.

3.2.1 VTP Concept

The goal of the *VTP* in *PyPLAS* for *Python programming* educations is to give students training opportunities of profoundly reading and analyzing *Python* codes. This problem type focuses on *code reading* because it plays an essential role in wring high quality codes for any programmer. An *VTP* instance consists of a *Python* source code, a set of questions with the answer forms, and the correct answers. A question asks answering the actual values of important variables or output messages when the source code is executed. The values are supposed to be displayed at the standard output in the code. Thus, to generate a new *VTP* instance, the corresponding standard output statements need to be added in the source code. The important variables and output messages to be traced are manually selected in our works. By carefully, reading and tracing the source code in each *VTP* instance, the learner can improve their knowledge of the corresponding study. Then, the design goals of the *VTP* are that:

1) a variety of useful source codes for *Python programming* study is depicted with full forms to novice learners,
2) a set of references describing the least *Python programming* topics related to the *VTP* instances are provided to assist novice learners,
3) learners can correctly answer the questions by carefully reading and understanding the source codes, and
4) any answer can be marked through string matching automatically.

3.2.2 Generation Procedure of VTP

A *VTP* instance can be generated by a teacher with the following procedure:

1) to select a source code that is suitable for the topic to be studied,
2) to find the important variables and messages to be traced in the code,
3) to add the corresponding standard output statements of them in the code,
4) to prepare the questions of asking the values/messages at the standard output and their correct answers,
5) to put together the source code, the questions, and the correct answers into one text file,
6) to run the program with the text file as the input that generates the *HTML/CSS/JavaScript* files for the offline answering function, and
7) to include the generated *VTP* instance in the assignment to students.

Using the implemented Java program and the Bash script, this procedure can be executed automatically.

3.2.3 Selection of Python Source Code

To clarify the generation procedure of *VTP*, I will explain the details by using the following **source code**. This source code intends for learners to understand the various output formatting structure of *Python programming*.

Listing 3.1: Source code example for VTP instance

```
# A quick-demo of Output formatting

x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))

#curly braces {} as placeholders and specify the order by using numbers (tuple index)
```

```
print('I love {0} and {1}'.format('bread', 'butter'))
print('I love {1} and {0}'.format('bread', 'butter'))

#keyword arguments to format the string
print('Hello {name}, {greeting}'.format(greating = 'Goodmorning', name = 'John
    '))
```

3.2.4 Generating Assignments

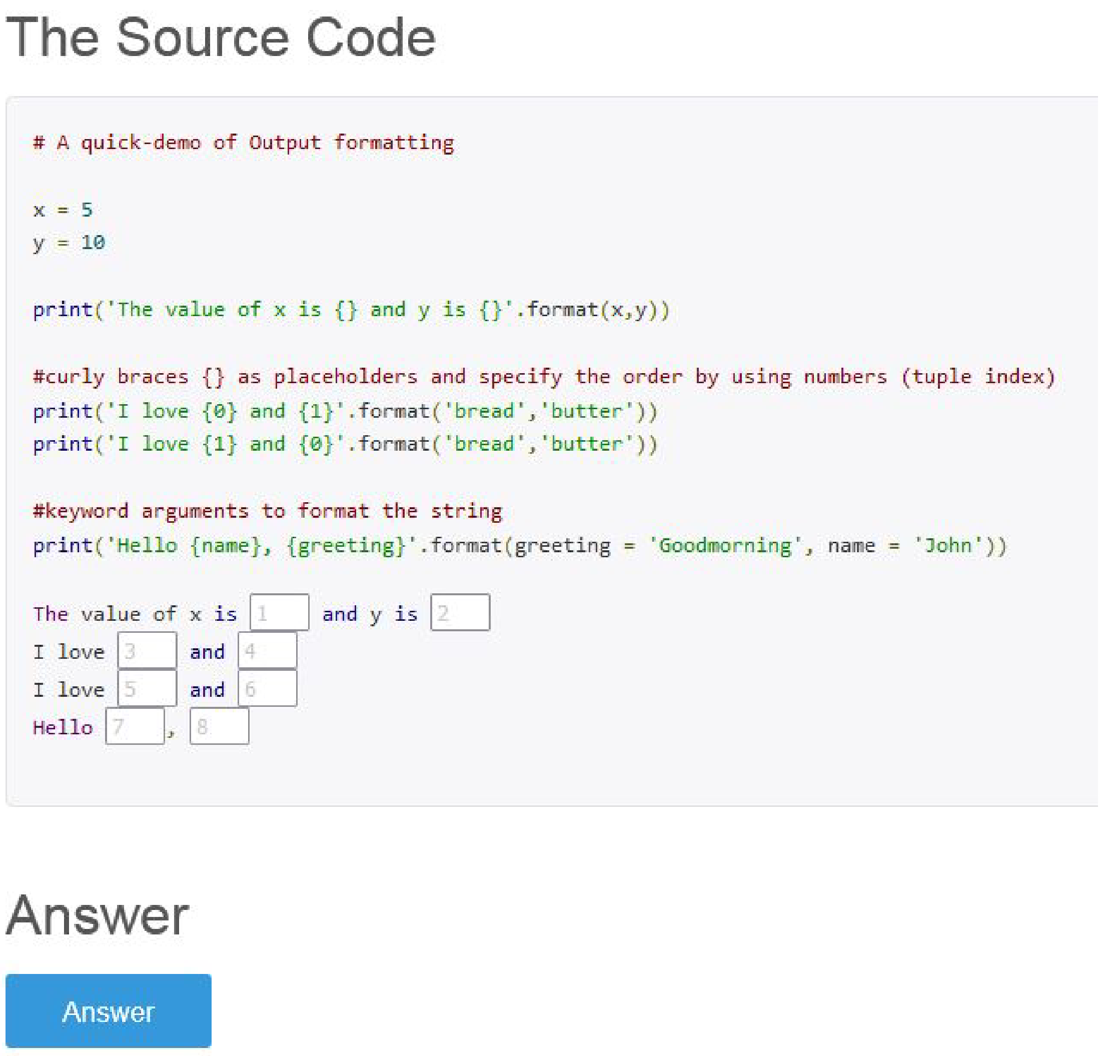
After selecting the source code, the teacher needs to add the corresponding standard output statements of them in the code, if necessary, and must prepare the questions of asking the values/messages at the standard output and their correct answers. Then, as the next step, the teacher needs to put together the source code, the questions, and the correct answers into one text file as shown in **listing 3.2**. Then, that file is passed as the input to run the program that generates the *HTML/CSS/-JavaScript* files for the offline answering function.

Listing 3.2: Input file example for VTP instance

```
# A quick−demo of Output formatting

x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))

#curly braces {} as placeholders and specify the order by using numbers (tuple index)
print('I love {0} and {1}'.format('bread', 'butter'))
print('I love {1} and {0}'.format('bread', 'butter'))

#keyword arguments to format the string
print('Hello {name}, {greeting}'.format(greating = 'Goodmorning', name = 'John
    '))

The value of x is _1_ and y is _2_
I love _3_ and _4_
I love _5_ and _6_
Hello _7_, _8_
5,10,bread,butter,butter,bread,John,Goodmorning
```

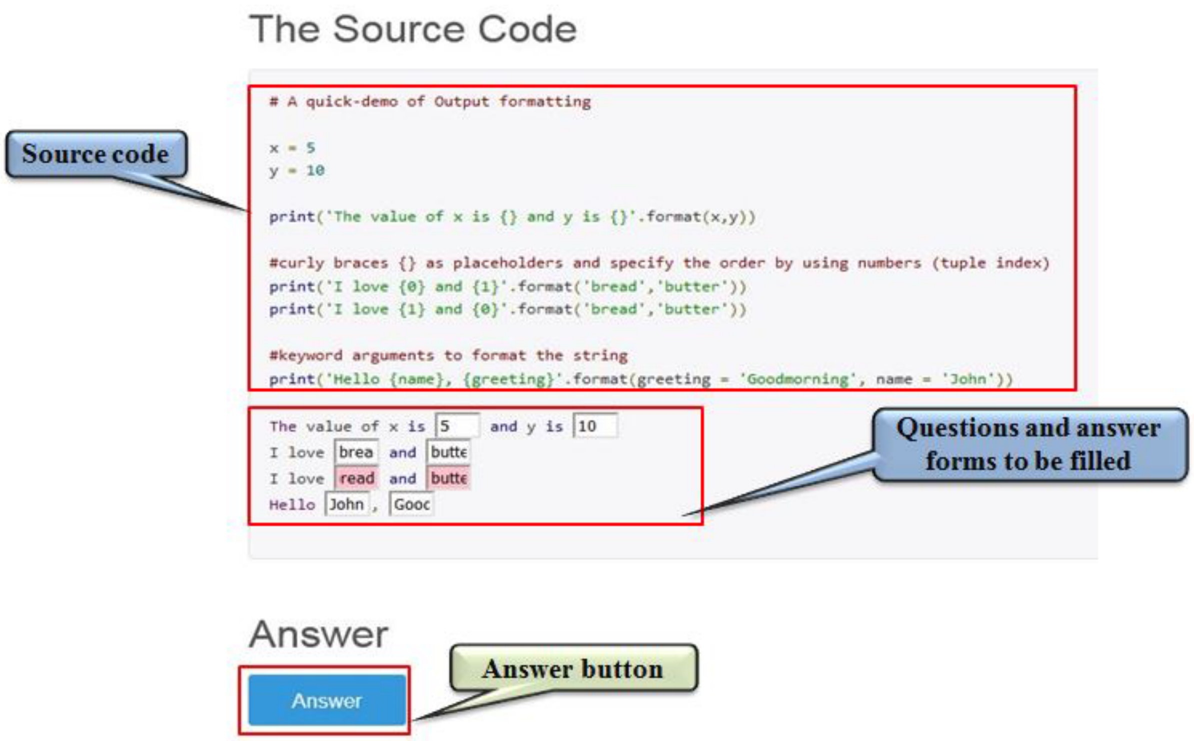
3.2.5 Answer Interface for VTP

The answer interface for the *VTP* instance is implemented on the web browser. It allows a student to solve *VTP* instances both on online and offline, since the answer marking is processed by running the JavaScript program on the browser. The correct answers to the questions are encrypted using SHA256 to avoid cheating by a student. Figure 3.1 illustrates the answer interface for an example *VTP* instance that will be seen by the students. It asks the four standard output messages for studying usages of various formatting styles in *Python programming*. The correct answers to them are 5, 10, bread, butter, butter, bread, John, and Goodmorning. After a student fills in the answer forms, he/she needs to click the "Answer" button. Depending on his/her answers, the form background will change to pink color if the answer is incorrect, otherwise, it will still remain the white color as shown in Figure 3.2.



```
# A quick-demo of Output formatting

x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))

#curly braces {} as placeholders and specify the order by using numbers (tuple index)
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))

#keyword arguments to format the string
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))

The value of x is [1] and y is [2]
I love [3] and [4]
I love [5] and [6]
Hello [7], [8]
```

Figure 3.1: VTP answer interface.



```
# A quick-demo of Output formatting

x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))

#curly braces {} as placeholders and specify the order by using numbers (tuple index)
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))

#keyword arguments to format the string
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
```

```
The value of x is 5 and y is 10
I love brea and butte
I love read and butte
Hello John , Gooc
```

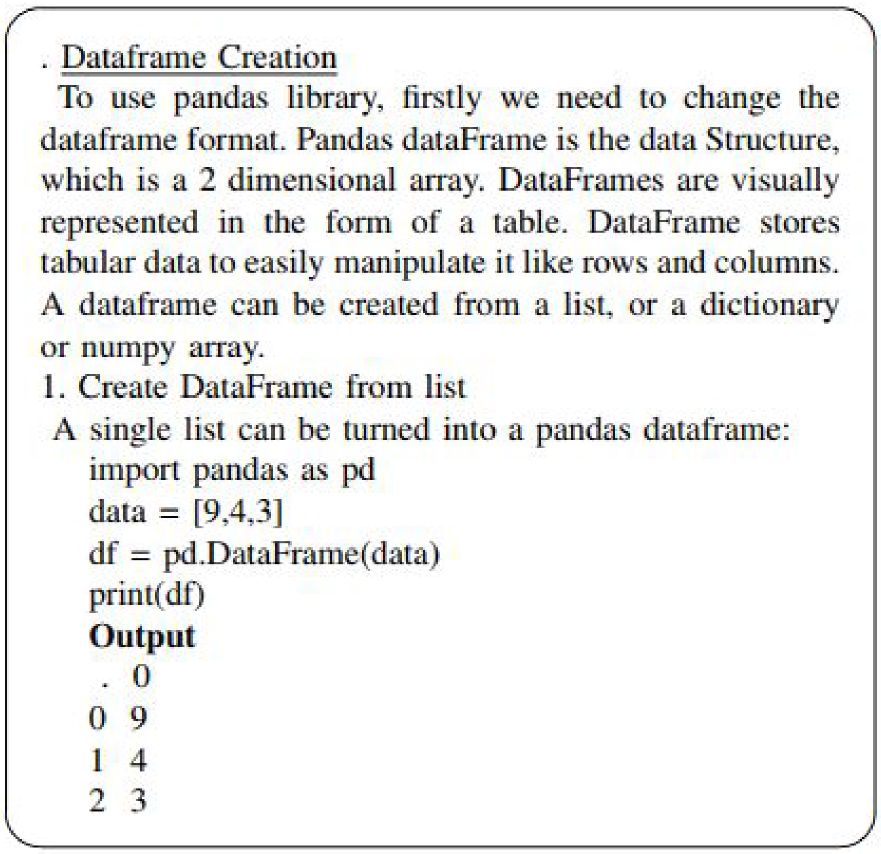
Figure 3.2: VTP answer interface with explanations in details.

3.2.6 Assisting References

To assist novice learners solving the *VTP instances* efficiently, simple reference documents are made by covering the least concepts necessary to solve the related *VTP* instances. By reading these references, learners can quickly access to the specific knowledge and information of *Python programming* of the current instance. Then, any learner is expected to easily solve the *VTP* instances, even if he/she has never studied *Python* programming. Table 3.1 shows the contents of the index page of the reference to solve the *VTP* instances on *pandas* library. This page shows the library concepts that will be studied in them, and their related *VTP* instance numbers. By clicking the link embedded at the concept, the related content page will appear. Figure 3.3 illustrates the content page on DataFrame Creation. It shows how to use the method .DataFrame() to create the data frame, the fundamental step of the pandas operations and it's corresponding output.

3.2.7 Overview of generated VTP instances

A total of 130 *VTP* instances are generated with the total of 805 answer forms. They are categorized into the four groups of basic grammar, advanced grammar, data structure or algorithm, and library usage. As common libraries, *numpy* and *pandas* are selected. The effectiveness of the *VTP* instances are evaluated through applications to 48 undergraduate students. Among them, 12 are the sixth-year students in a Myanmar university who have studied the *Python programming* for two years, and 36 are the third-year students in an Indonesia university who have never studied it.


. Dataframe Creation

To use pandas library, firstly we need to change the dataframe format. Pandas dataFrame is the data Structure, which is a 2 dimensional array. DataFrames are visually represented in the form of a table. DataFrame stores tabular data to easily manipulate it like rows and columns. A dataframe can be created from a list, or a dictionary or numpy array.

1. Create DataFrame from list

A single list can be turned into a pandas dataframe:

```
import pandas as pd
data = [9,4,3]
df = pd.DataFrame(data)
print(df)
```

Output

```
 .  0
0  9
1  4
2  3
```

Figure 3.3: Content page of DataFrame creation.

Table 3.1: Index page of reference on pandas library.

Pandas Concepts	Related VTPs Version	Related Problem Numbers
dataframe creation	VTP-5	1
indexing and selecting data	VTP-5	6
excel file reading	VTP-5	2
excel file writing	VTP-5	3
what is CSV file?	VTP-5	4, 5
iterate pandas dataframe	VTP-5	7
aggregration in pandas dataframe	VTP-5	8
pandas datetime function	VTP-5	9
options and customizations	VTP-5	10

3.3 VTP for Basic Grammar

First, we discuss the *VTP* instances for basic grammar category. They cover standard input/output, operator usages, data types, control flows, and so on as base of *Python programming*.

3.3.1 Learning Objectives

Learning the basic grammar of *Python* is essential to understand its own set of rules and syntax which is crucial for writing correct and functional programs. Also, it can improve code readability, error identification and debugging process to identify and fix syntax errors in the code. Learning the basic grammar of *Python* provides the learners with a strong foundation, and enables the learners to advance programming skills. Any leaner must master these concepts at the first-step study of *Python programming*.

3.3.2 Generated VTP instances

For this category, 47 instances are generated using *Python* source codes in [5]. Table 3.2 shows the topic, the number of lines of the source code, the number of questions, the number of answer forms, and the average correct answer rate by students for each instance.

3.3.3 Solution Results

The average correct rate in Table 3.2 by the students is 90% or higher for any instance, and the average one among all the instances is 96.5%. Many students have no experience of studying *Python* programming. Thus, the *VTP* instances in the basic grammar category have proper difficulty levels for novice learners. As the summary of the solution results, Table 3.3shows the solution levels, the range of the number of correctly answered forms, the number of students, the range of the number of instances attempted to solve, and the average number of answer submissions with its standard deviation (SD). 48 students in groups A and B submitted their answers 2.3 times on average for each instance. It is noted that at least 47 submissions are necessary to solve 47 instances. One student in group C gave up solving the *VTP* instances after attempting four instances.

3.4 VTP for Advanced Grammar

Second, we discuss the *VTP* instances for the advanced grammar category. They cover object oriented programming concepts, arrays with various functions, JSON file usage, and other advanced topics of *Python programming*. JSON is used at transmitting structured data over networks, and is very popular in programming, although it is not familiar to novice learners.

3.4.1 Learning Objectives

Learning advanced grammar in *Python* builds upon the foundation of basic grammar and it can expand the programming skills, enhance code efficiency, and gain the ability to tackle more complex programming tasks. It also enables to write cleaner, more maintainable code, opening up opportunities for advanced projects and collaborations. In additions, understanding advanced grammar is crucial for effectively utilizing *Object Oriented Programming (OOP)* concepts such as classes, objects, inheritance, polymorphism, and encapsulation. OOP allows for creating modular and reusable code, leading to better software design and maintainability. Moreover, it provides robust mechanisms for handling exceptions to gracefully handle errors and exceptions in the code, improving its reliability and stability.

Therefore, as the overall, learning advanced grammar in *Python* empowers for writing more efficient, modular, and powerful code. It expands the capabilities as a *Python* developer and enables to tackle complex problems and build sophisticated applications.

3.4.2 Generated VTP instances

For this category, 17 instances in Table 3.4 are generated using source codes in [5].

3.4.3 Solution Results

For the *VTP* instances in this category, 38 students submitted their answers. Then, the average correct rate in Table 3.4 is 94% or higher for any instance except for JSON data reading with ID=13. The average correct rate of all the instances is 96.5%. The JSON data reading topic can be hard for students because they have little knowledge on syntax structure of data reading from JSON file. The improvement for this topic will be in future works.

The summary results in Table 3.5 show that 34 students in groups A and B correctly answered at 100 or more forms among 108. For each instance, 24 students in group A submitted their answers 2.8 times and 10 in group B submitted 6.1 times on average.

3.5 VTP for Data Structures and Algorithms

Third, we discuss the *VTP* instances for data structure and algorithm category. They cover stack, queue, sorting, searching, graph, Dijkstra algorithm, and other related topics.

3.5.1 Learning Objectives

Data structures and algorithms play crucial roles in designing modular and reusable codes. By organizing data in appropriate data structures and implementing efficient algorithms, we can create code components that can be easily reused across different projects. When working with existing codebases or using third-party libraries, having a good understanding of data structures and algorithms allows the any developer to comprehend the underlying implementation and make informed decisions on how to best utilize those libraries. In addition, by understanding different data structures like arrays, linked lists, stacks, queues, trees, graphs, and algorithms like sorting, searching, and traversals, we can easily gain the ability to analyze problems, design algorithms, and implement solutions.

In summary, learning *Python* data structures and algorithms enables to manipulate data efficiently, solve problems effectively, optimize code performances, design modular code, and excel in technical interviews. It forms the backbone of efficient programming and is essential for any developer or data scientist working with *Python*. Any learner should know well these concepts to write proper codes for handling data effectively. Therefore, this category is introduced at the third-step study of *Python programming*.

3.5.2 Generated VTP instances

For this category, 24 instances in Table VI are generated using source codes in [5].

3.5.3 Solution Results

For the *VTP* instances in this category, 25 students submitted answers. The average correct rate in Table 3.6 is 90% or higher for any instance, except for the six instances, where the infix to postfix conversion with ID=6 is only 79%. The average correct rate of all the instances is 92.4%, which is smaller than the rates for previous categories. The topics in this category can be hard for students because they do not have enough knowledge on data structure and algorithm. The improvements for these topics will be in future works. The summary results in Table 3.7 shows that 11 students in group A correctly answered all the forms and submitted answers 1.7 times for each instance on average. 13 students in group B correctly answered 160 or more forms among 187 and submitted answers 3.3 times on average. One student in group C gave up solving most *VTP* instances after attempting five ones.

3.6 VTP for Numpy Library

Fourth, we discuss the *VTP* instances for numpy library usage to study the fundamental package for scientific computing and machine learning.

3.6.1 Learning Objectives

The *numpy* library is an essential tool for scientific computing and data analysis in *Python*. It provides powerful data structures, numerical computing tools, and a collection of mathematical functions to efficiently work with large arrays and matrices. Some of the functions that can be provided by *numpy* library are summarized as follows:

1. Efficient array operations: the main feature of *numpy* is the *ndarray* (n-dimensional array) object, which allows for performing operations on large arrays of homogeneous data efficiently.

2. Mathematical functions and tools: *numpy* provides a wide range of mathematical functions, including trigonometry, linear algebra, statistics, random number generation, and more.

3. Integration with other libraries: *numpy* is a foundational library in the *Python* scientific computing ecosystem. It integrates well with other libraries such as s*cipy*, *pandas*, *matplotlib*, and *scikit-learn*.

4. Vectorized operations: *numpy* allows for performing on entire arrays or large subsets of data without writing explicit loops.

5. Interoperability with other languages: *numpy* arrays can be easily shared with other libraries and languages like *C/C++, Fortran, and MATLAB*. This makes it convenient to leverage existing code and tools from different domains.

6. Data analysis and manipulation: *numpy* provides various functionalities for data manipulation, such as reshaping, indexing, slicing, merging, and sorting. These operations are crucial for data preprocessing and analysis tasks.

Overall, studying the *numpy* library is important for anyone working with data, scientific computing, or numerical analysis in *Python*. It offers efficient array operations, mathematical functions, memory optimization, integration with other libraries, and tools for data manipulation. By mastering *numpy*, the learner can significantly enhance the productivity and tackle complex computational tasks more effectively. Therefore, we focus to generate *VTP* instances to this topic.

3.6.2 Generated VTP instances

For this category, 32 instances in Table 3.8 are generated using source codes in [5]

3.6.3 Solution Results

For the *VTP* instances in this category, 25 students submitted answers. The average correct rate in Table 3.8 is 90% or higher for any instance, except for the six instances on array creation with arrange method, shuffling usage, and complex function usage. Among them, shuffling strings with ID=21 gives only 74%. The average correct rate of all the instances is 92.2%, which is similar to the rate in the data structure and algorithm category. As the summary of the solution results, Table 3.9 shows that two students in group A correctly answered all the forms and submitted answers 2.3 times for each instance on average. 21 students in group B answered 150 or more forms among 175 and submitted answers 2.9 times on average. One student in group D gave up solving the *VTP* instances after attempting six ones.

3.7 VTP for Pandas Library

Last, we discuss the *VTP* instances for the *pandas* library usage by covering data frame creation, excel and csv files reading and writing, indexing and selecting data, iteration on pandas, date time creation and so on.

3.7.1 Learning Objectives

The *pandas* library is a powerful tool for data manipulation, analysis, and cleaning. It provides a high-performance, easy-to-use data structure called *DataFrame*, which is designed to handle structured data efficiently. This library has been most widely used in data science, data analysis, and machine learning. Some operations that are supported by the *pandas* library are summarized as follows.

1. Data handling and manipulation: *pandas* simplifies the process of loading, manipulating, and cleaning data. It offers a wide range of data manipulation functions and methods, such as filtering, sorting, grouping, merging, reshaping, and pivoting to efficiently transform and prepare data for analysis.

2. Tabular data representation: The *DataFrame* data structure in *pandas* provides a tabular representation of data, similar to a spreadsheet or a SQL table for organizing, analyzing, and manipulating data with rows and columns with structured data.

3. Data analysis and exploration: *pandas* provides a rich set of functions for descriptive statistics, data summarization, and exploration. Additionally, pandas integrates with other libraries like *numpy* and *matplotlib*, allowing for seamless data analysis and visualization workflows.

4. Missing data handling: *pandas* offers flexible tools to handle missing values, allowing to drop or fill missing data based on the requirements to perform data analysis accurately and effectively.

5. Time series analysis: *pandas* has extensive support for analyzing and working with time-dependent data. It provides functionalities to handle and manipulate time-stamped data, resample data at different frequencies, perform date/time calculations, and handle time zone conversions.

6. Data input/output (I/O): *pandas* supports reading and writing data from various file formats, such as CSV, Excel, SQL databases, and more. It simplifies the process of importing and exporting data, making it easier to work with different data sources and integrate with other tools and systems.

7. Efficiency and performance: *pandas* is built on top of *numpy*, which allows for efficient and optimized data operations. It provides vectorized operations, which are much faster than traditional loops, resulting in improved performance when working with large datasets.

In summary, studying and using the *pandas* library is important for anyone working with data in *Python*. It offers a comprehensive set of tools for data manipulation, analysis, and cleaning, simplifying the data wrangling process. The *pandas* library provides a flexible and efficient way to handle structured data, perform data analysis, and prepare data for further processing or visualization. Therefore, since every *Python programming* learner should know how to use this library, we introduced the *VTP* instances about this concept.

3.7.2 Generated VTP instances

For this category, 10 instances in Table 3.10 are generated using source codes in [5].

3.7.3 Solution Results

For the *VTP* instances in this category, 15 students submitted answers. The average correct rate in Table 3.10 is 98% or higher for any instance, and the average rate for all the instances is 99.7%. All the students can solve any instance correctly except for two instances with ID=6 and ID=7.

The summary results in Table 3.11 show that 11 students in group A correctly answered 47 or 48 forms among the 48, and four students in group B correctly answered at 31 or more. The average number of answer submissions is 1.2 and 2.5 for each instance, respectively.

3.8 Summary

In this chapter, we present the five categories of the *Value Trace Problem (VTP)* in PyPLAS with 130 instances by collecting a set of source codes from textbooks, and websites. Besides, assisting references on *Python programming* topics related to the *VTP* instances were introduced to assist novice learners in solving them effectively. Then, all of the problem instances are assigned to 48 undergraduate students in Myanmar, and Indonesia. Then, we analyzed the student's solution results including their submission times to know individual student's performances, generated problem instances' difficulties, and to confirm the effectiveness as the viable self-study learning tool. The future studies include the generation of value trace problem for the remaining categories to cover other useful libraries for machine learning, deep learning, mutimedia, and so on.

Table 3.2: VTP instances for basic grammar.

ID	basic grammar concept	# of lines	# of questions	# of forms	avg correct rate
1	standard input/output	6	3	3	99%
2	formatting usage	6	4	8	98%
3	implicit type	12	9	9	94%
4	explicit type	9	5	5	94%
5	arithmetic operators	8	6	6	98%
6	comparison operators	7	5	5	98%
7	logical operators	5	3	3	98%
8	identity operators	9	3	3	98%
9	membership operator	6	4	4	98%
10	various dataTypes	29	18	27	93%
11	sep and end keyword	14	1	20	91%
12	if else statement	9	2	2	98%
13	if-elseif-else	9	1	18	98%
14	nested-if-else	7	1	2	98%
15	for loop usage	5	1	4	98%
16	for loop with range	4	1	1	90%
17	while loop usage	4	1	3	98%
18	while-else usage	6	1	5	98%
19	nested while loop	7	1	6	95%
20	break statement	6	1	5	97%
21	continue statement	4	1	3	98%
22	pass statement usage	11	1	6	94%
23	accessing string	6	1	5	98%
24	slicing operation	6	5	5	93%
25	string concatenation	7	2	2	92%
26	string replication	4	2	2	98%
27	string membership	7	3	3	98%
28	relational operator	6	2	2	98%
29	tuple usage	7	1	6	98%
30	negative tuple index	3	1	2	97%
31	nested tuple usage	3	1	2	98%
32	changing tuple elements	4	1	10	98%
33	slicing on tuple	6	1	15	97%
34	tuple membership	6	1	10	91%
35	iteration on tuple	3	1	4	98%
36	list usage	7	1	6	97%
37	slicing on list	5	1	16	98%

ID	basic grammar concept	# of lines	# of questions	# of forms	avg correct rate
38	negative index usage	4	1	3	97%
39	addition on list	7	1	16	91%
40	updating list elements	5	1	8	97%
41	deletion on list	6	1	8	98%
42	three deletion methods	7	1	10	97%
43	dictionary usage	3	2	2	94%
44	changing dict elements	7	4	4	98%
45	adding dictionary entry	4	2	2	98%
46	loop through dictionary	3	6	6	98%
47	deletion on dictionary	8	1	7	98%
average		6.7	2.5	6.1	96.5%
total		317	117	287	SD: 2.4%

Table 3.3: Solution results for basic grammar.

level group	range of # correctly solved forms	# of students	# of attempted instances	ave. # of submissions (SD)
A	287	23	47	109.2 (51.0)
B	286-250	24	47-46	109.9 (33.2)
C	12	1	4	17

Table 3.4: VTP instances for advanced grammar.

ID	advanced grammar concept	# of lines	# of questions	# of forms	avg correct rate
1	object creation	8	1	3	99%
2	default constructor	7	1	1	97%
3	parameterized constructor	16	1	3	100%
4	object's information	15	6	8	100%
5	array creation	10	2	6	97%
6	adding array items	21	4	14	100%
7	accessing array items	7	4	4	100%
8	removing array items	16	4	10	98%
9	slicing array items	16	3	8	97%
10	searching array items	10	3	8	100%
11	sep and end keyword	14	3	12	99%
12	if else statement	5	1	5	94%
13	if-elseif-else	10	1	3	75%
14	nested-if-else	27	1	6	96%
15	for loop usage	10	1	12	97%
16	for loop with range	7	1	2	95%
17	while loop usage	6	1	3	97%
average		12.1	2.2	6.4	96.5%
total		205	38	108	SD: 5.9%

Table 3.5: Solution results for advanced grammar.

level group	range of # correctly solved forms	# of students	# of attempted instances	ave. # of submissions (SD)
A	108	24	17	48.1 (32.8)
B	107-100	10	17-16	103.8 (2.0)
C	99-81	4	17	93.3 (8.3)

Table 3.6: VTP instances for data structures & algorithms.

ID	data structure or algorithm	# of lines	# of questions	# of forms	avg correct rate
1	stack operations	25	7	7	100%
2	parentheses checker	22	1	7	89%
3	symbols checker	29	1	7	98%
4	decimal to binary	16	8	8	88%
5	decimal to any base	17	4	4	84%
6	infix-to-postfix	33	1	6	79%
7	postfix evaluation	25	4	4	84%
8	queue operations	24	7	7	96%
9	deque operations	28	5	5	96%
10	palindrome checker	16	1	8	92%
11	bubble sort	14	1	13	91%
12	selection sort	12	1	12	92%
13	insertion sort	14	1	12	96%
14	shell sort	19	2	12	92%
15	merge sort	31	10	11	96%
16	quick sort	31	1	12	96%
17	sequential search	13	1	9	96%
18	binary search	19	6	10	96%
19	hash table usage	64	1	4	92%
20	graph implementation	56	1	12	86%
21	breadth first search	28	1	4	96%
22	depth first search	24	1	4	96%
23	Dijkstra algorithm	40	1	5	92%
24	Prim algorithm	39	4	4	96%
average		26.6	3.0	7.8	92.4%
total		639	71	187	SD: 5.1%

Table 3.7: Solution results for data structure algorithms.

level group	range of # correctly solved forms	# of students	# of attempted instances	ave. # of submissions (SD)
A	187	11	24	41.7 (17.1)
B	184-160	13	24-21	73.7 (55.1)
C	32	1	5	93.3 (50)

Table 3.8: VTP instances for *numpy* library.

ID	*numpy* library concepts	# of lines	# of questions	# of forms	avg correct rate
1	numpy array creation	23	8	9	91%
2	basic operations	12	7	10	96%
3	array characteristics	7	5	5	95%
4	arranging array elements	6	4	4	86%
5	reshaping array elements	9	4	6	99%
6	index arrays	8	3	3	93%
7	basic slicing usage	8	3	4	96%
8	boolean indexing	24	5	4	96%
9	iterating over array	14	4	18	93%
10	modifying array elements	10	3	3	88%
11	broadcasting iteration	13	1	9	91%
12	bitwise operations	19	3	3	96%
13	rounding function	13	3	5	96%
14	arithmetic functions	27	6	9	95%
15	complex functions	9	2	2	88%
16	string operations	9	6	6	91%
17	accessing string	10	6	6	93%
18	string comparisons	19	7	8	96%
19	shuffling usage	13	3	7	85%
20	shuffling two lists	14	3	4	90%
21	shuffling strings	7	2	2	74%
22	dot operations	11	3	4	96%
23	maximum operation	19	3	3	92%
24	minimum operation	19	3	3	93%
25	arithmetic mean	10	4	4	96%
26	variance operation	13	3	3	82%
27	standard deviation	10	4	4	91%
28	vertically stacking	13	1	2	91%
29	append function	17	4	10	94%
30	sorted function	8	2	4	96%
31	argument sorting	9	2	4	96%
32	lexsort usage	8	1	7	96%
average		12.8	3.7	5.5	92.2%
total		411	118	175	SD: 5.1%

Table 3.9: Solution results for *numpy* library.

level group	range of # correctly solved forms	# of students	# of attempted instances	ave. # of submissions (SD)
A	175	2	32	75
B	174-150	21	32-31	92 (47.7)
C	130	1	32	53.7 (64)
D	30	1	6	93.3 (55)

Table 3.10: VTP instances for pandas library.

ID	pandas library concept	# of lines	# of questions	# of forms	avg correct rate
1	data frame creation	12	1	6	100%
2	excel file reading	8	1	5	100%
3	excel file writing	11	1	5	100%
4	csv file reading	15	1	3	100%
5	csv file writing	10	1	3	100%
6	index and select data	16	1	9	99%
7	iteration on pandas	8	1	8	98%
8	aggregation on pandas	7	1	3	100%
9	date time creation	10	1	4	100%
10	pandas options	12	2	2	100%
average		10.9	1.1	4.8	99.7%
total		100	11	48	SD: 0.7%

Table 3.11: Solution results for pandas library.

level group	range of # correctly solved forms	# of students	# of attempted instances	ave. # of submissions (SD)
A	48-47	11	10	12 (1.4)
B	46-31	4	10	25 (16.8)

Chapter 4

Code Modification Problem

In this chapter, we present the overview of *Code Modification Problem (CMP)*, its generation procedure, and example of *CMP* instance generation in details with two level answer marking in *Python Programming Learning Assistant System (PyPLAS)*.

4.1 Introduction

In the previous chapter, we discussed *VTP* in *PyPLAS* to offer code reading study of various grammar concepts and common libraries. A *VTP* instance asks a learner to trace the actual values of the important variables or output messages in the given source code. Unfortunately, VTP and any type of exercise problems in *PyPLAS* may not suitable to study some *Python* topics that need the illustrating figures such as bar charts, line charts, pie charts, and so on. Therefore, we introduce the *code modification problem (CMP)* for self-study of *Python programming*, as a new problem type for *PyPLAS*.

The rest of this chapter is organized as follows. Firstly, the overview of *CMP* is introduced. Then, the next section will present the step by step generation procedure of *CMP* instance. After that, it will show the example of *CMP* instance generation with appropriate source code selection, user interface, and so on. Then, it will review the two level answer marking function. Finally, the conclusion is given for this chapter.

4.2 Overview of Code Modification Problem (CMP)

In the *CMP* instance, basically, one source code, and two images are shown to the learner. The first image shows the web page that will be obtained by running the source code. Then, the learner will need to modify the source code so that it can generate the second image. The learner is requested to carefully check the differences between the two images for the correct answers. The correctness of the student answer is verified through *string matching* at each statement in the answer to the corresponding original statement in the source code. The design goals of *CMP* are described by:

1. A variety of useful and practical source codes for studying *Python* topics is depicted with full forms to novice earners,

2. A learner can correctly answer the questions of *CMP* by carefully reading the source code and modifying the required parts in the functions, variables, and parameters.

3. Any answer can be marked through string matching automatically.

4.3 CMP Instance Generation Procedure

In this section, the generation procedure of the new *code modification problem (CMP)* is presented. An instance of *CMP* can be generated through the following steps:

1. To select a source code from a website or a text book that is suitable for the current topic,
2. To find the important parts (functions, variables, parameters) in the source code to be modified so that the learner can study the instance topic,
3. To prepare another source code by replacing the modified parts from the original source code,
4. To put together the source code, the modified source code, and the correct answers into one text file,
5. To run the answer interface generation program with the text file as the input to generate the *CMP* instance with *HTML/CSS/JavaScript* files for the offline answering function,
6. To obtain the two images by running the original and modified source codes and add them into the generated *CMP* instance, and
7. To assign the generated *CMP* instance to students.

By using the Java programs and the Bash script, this procedure can be executed automatically.

4.4 Example of CMP Instance Generation

In this section, we present the details of each step for the *CMP* instance generation using a sample source code.

4.4.1 Source Code Selection

A proper source code should be selected to be studied by the students. Here, the source code in Listing 6.1 is selected for further discussions. This source code uses the donut chart and its related functions.

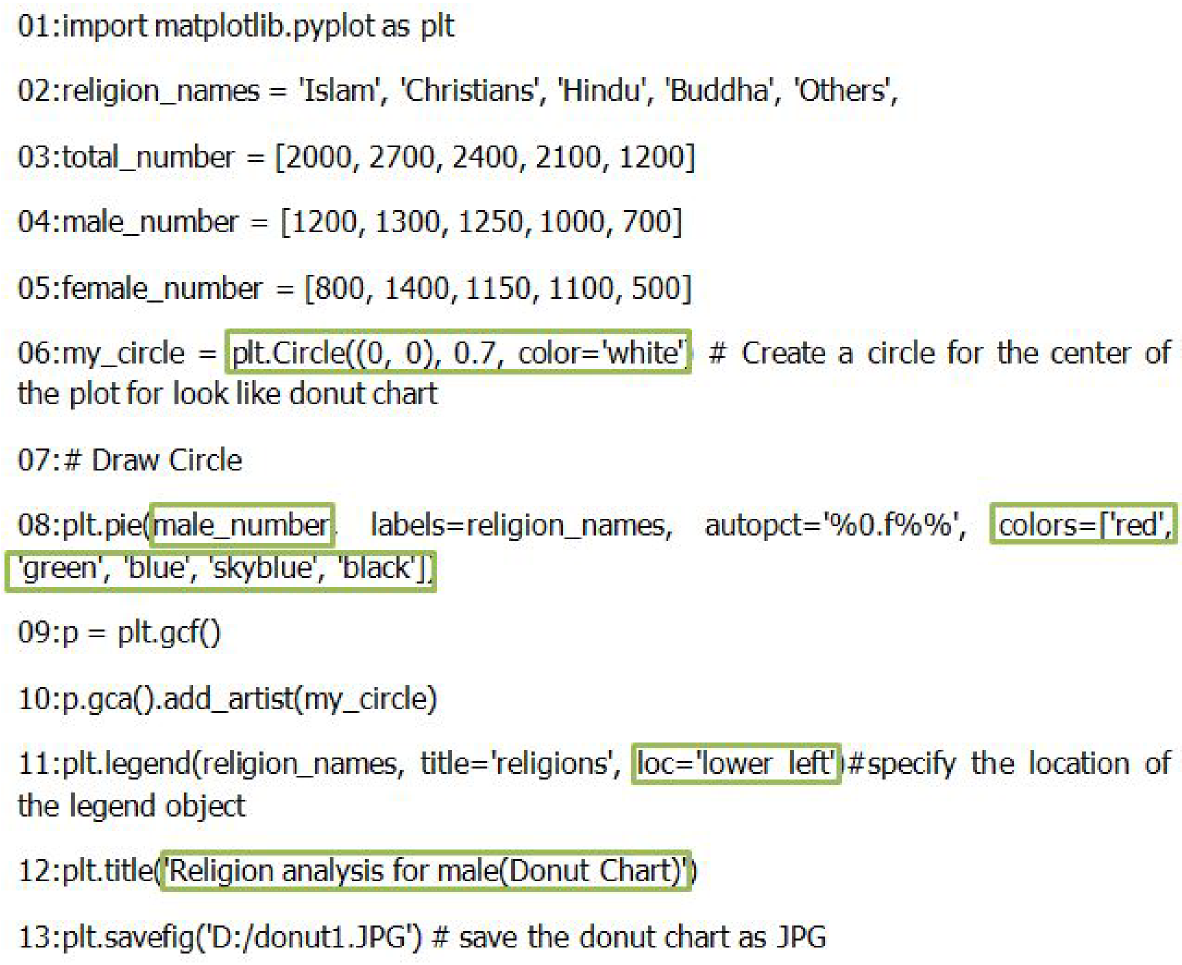
Listing 4.1: Source code example for CMP instance

```
import matplotlib.pyplot as plt
religion_names = 'Islam', 'Christians', 'Hindu', 'Buddha', 'Others',
total_number = [2000, 2700, 2400, 2100, 1200]
male_number = [1200, 1300, 1250, 1000, 700]
female_number = [800, 1400, 1150, 1100, 500]
my_circle = plt.Circle((0, 0), 0.7, color='white')# Create a circle for the center of the plot
    for look like donut chart
# Draw Circle
plt.pie(male_number, labels=religion_names, autopct='%0.f%%', colors=['red', 'green'
    , 'blue', 'skyblue', 'black'])
p = plt.gcf()
```

```
10 p.gca().add_artist(my_circle)
11 plt.legend(religion_names,title='religions', loc='lower left')#specify the location of the legend object
12 plt.title('Religion analysis for male(Donut Chart)')
13 plt.savefig('D:/donut1.JPG') # save the donut chart as JPG
```

4.4.2 Finding Important Parts in Source Code

After selecting the appropriate source code, the teacher needs to find the important parts (functions, variables, parameters) in the source code such that they should be studied and understood clearly by the students. Figure 4.1 illustrates the important parts that will significantly affect the output results in the above source code to be modified.



```
01:import matplotlib.pyplot as plt
02:religion_names = 'Islam', 'Christians', 'Hindu', 'Buddha', 'Others',
03:total_number = [2000, 2700, 2400, 2100, 1200]
04:male_number = [1200, 1300, 1250, 1000, 700]
05:female_number = [800, 1400, 1150, 1100, 500]
06:my_circle = plt.Circle((0, 0), 0.7, color='white') # Create a circle for the center of the plot for look like donut chart
07:# Draw Circle
08:plt.pie(male_number, labels=religion_names, autopct='%0.f%%', colors=['red', 'green', 'blue', 'skyblue', 'black'])
09:p = plt.gcf()
10:p.gca().add_artist(my_circle)
11:plt.legend(religion_names, title='religions', loc='lower left')#specify the location of the legend object
12:plt.title('Religion analysis for male(Donut Chart)')
13:plt.savefig('D:/donut1.JPG') # save the donut chart as JPG
```

Figure 4.1: Source code with highlighting important parts.

4.4.3 Preparing Another Source Code

Then, as the third step, the teacher has to prepare another source code by replacing the important parts in the original source code that are previously mentioned in Section 4.4.2. **L**isting 4.2 illustrates the newly created source code with the modifications of line 6, line 8, line 11, and line 12 that are needed to be solved by the students.

Listing 4.2: Modified source code as output code

```
import matplotlib.pyplot as plt
religion_names = 'Islam', 'Christians', 'Hindu', 'Buddha', 'Others',
total_number = [2000, 2700, 2400, 2100, 1200]
male_number = [1200, 1300, 1250, 1000, 700]
female_number = [800, 1400, 1150, 1100, 500]
my_circle = plt.Circle((0, 0), 0.7, color='gray')# Create a circle for the center of the plot
    for look like donut chart
# Draw Circle
plt.pie(female_number, labels=religion_names, autopct='%0.f%%', colors=['black', '
    skyblue', 'blue', 'green', 'red'])
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.legend(religion_names,title='religions', loc='upper right')#specify the location
    of the legend object
plt.title('Religion analysis for female(Donut Chart)')
plt.savefig('D:/donut2.JPG') # save the donut chart as JPG
```

4.4.4 Input File for Generating CMP Instance

Then, as the next step, the teacher needs to put together the source code, the modified source code, and the correct answers into one text file with the appropriate format to satisfy the required specifications from the *CMP* generator and pass it as the input file to run the *CMP* generation program to generate the *CMP* instance with the *HTML/CSS/JavaScript* files for the answering interface. For checking the students' results, we implement two level answer marking that will be discussed in Section 4.5.

4.4.5 Adding Two images

After generating a *CMP* instance from the generator, the teacher needs to add two images obtained by running the original and modified source codes into the generated *CMP* instance so that the learner can carefully read the source code to know difference between the two images for the correct answer.

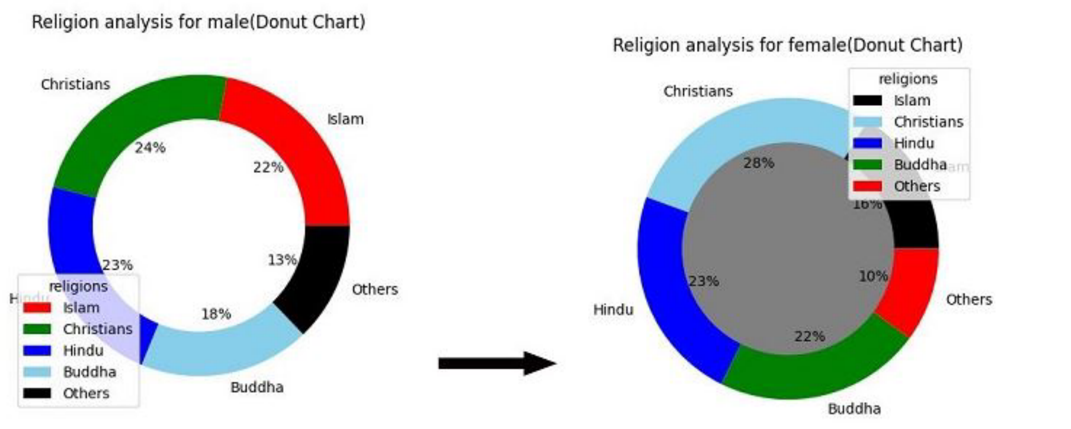
4.4.6 Problem Answer Interface

The answer interface for a *CMP* instance is implemented on the web browser. It allows a learner to solve the *CMP* instance both on online and offline, since the answer marking is applied by running the *JavaScript* program on the browser. The correct answers to the questions are encrypted using SHA256 to avoid cheating by a student.

Figure 4.2 illustrates the user interface design of the example *CMP* instance on a web browser. "Source Code" shows the problem source code of the *CMP* instance. "the output" shows the answer forms where each line corresponds to one statement in the problem source code that may need to modify. By understanding the given source code and the first image, the learner is requested to modify the corresponding lines in the source code of the output area to obtain the second image by running the modified code. After the modification, the "Answer" button should be clicked. If the answer is not correct, the background color of the corresponding input form becomes pink, otherwise, it is white that can be seen in Figure 4.3. It is possible to submit answers repeatedly until all the answers become correct.

Problem #10

Please modify the following source code in the output field to change from the left figure to the right figure!



Click Here To Download Source Code

Source Code

```
import matplotlib.pyplot as plt
religion_names = 'Islam', 'Christians', 'Hindu', 'Buddha', 'Others',
total_number = [2000, 2700, 2400, 2100, 1200]
male_number = [1200, 1300, 1250, 1000, 700]
female_number = [800, 1400, 1150, 1100, 500]
my_circle = plt.Circle((0, 0), 0.7, color='white')# Create a circle for the center of the plot for look like donut chart
# Draw Circle
plt.pie(male_number, labels=religion_names, autopct='%0.f%%', colors=['red', 'green', 'blue', 'skyblue', 'black'])
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.legend(religion_names,title='religions', loc='lower left')#specify the location of the legend object
plt.title('Religion analysis for male(Donut Chart)')
plt.savefig('D:/donut1.JPG') # save the donut chart as JPG
```

the output

```
import matplotlib.pyplot as plt
religion_names = 'Islam', 'Christians', 'Hindu', 'Buddha', 'Others',
total_number = [2000, 2700, 2400, 2100, 1200]
male_number = [1200, 1300, 1250, 1000, 700]
female_number = [800, 1400, 1150, 1100, 500]
my_circle = plt.Circle((0, 0), 0.7, color='white')# Create a circle for the center of the plot for look like donut chart
# Draw Circle
plt.pie(male_number, labels=religion_names, autopct='%0.f%%', colors=['red', 'green', 'blue', 'skyblue', 'black'])
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.legend(religion_names,title='religions', loc='lower left')#specify the location of the legend object
plt.title('Religion analysis for male(Donut Chart)')
plt.savefig('D:/donut1.JPG') # save the donut chart as JPG
```

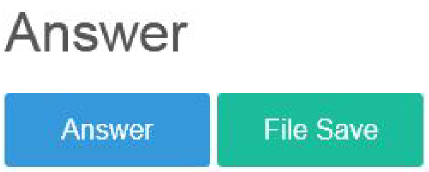
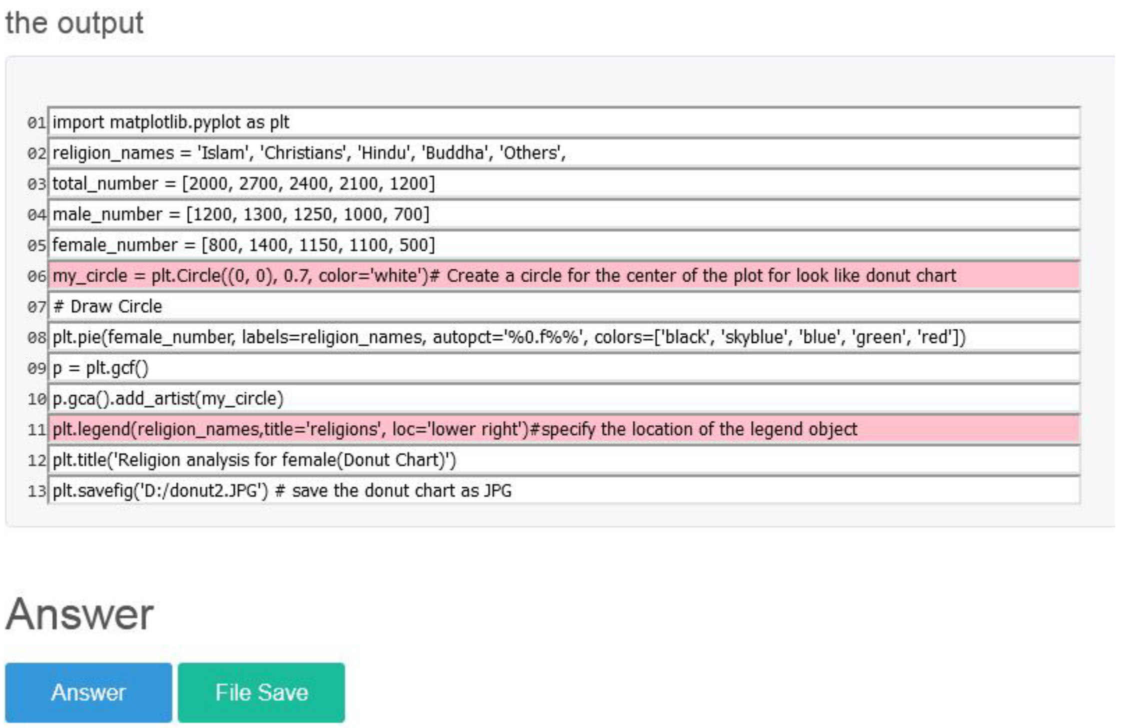


Figure 4.2: CMP user interface.



```
import matplotlib.pyplot as plt
religion_names = 'Islam', 'Christians', 'Hindu', 'Buddha', 'Others',
total_number = [2000, 2700, 2400, 2100, 1200]
male_number = [1200, 1300, 1250, 1000, 700]
female_number = [800, 1400, 1150, 1100, 500]
my_circle = plt.Circle((0, 0), 0.7, color='white')# Create a circle for the center of the plot for look like donut chart
# Draw Circle
plt.pie(female_number, labels=religion_names, autopct='%0.f%%', colors=['black', 'skyblue', 'blue', 'green', 'red'])
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.legend(religion_names,title='religions', loc='lower right')#specify the location of the legend object
plt.title('Religion analysis for female(Donut Chart)')
plt.savefig('D:/donut2.JPG') # save the donut chart as JPG
```

Figure 4.3: CMP user interface with error messages.

4.5 Two-Level Answer Marking

When a student clicks the answer button in the interface, the two-level marking is applied for marking the student answers. The answer is marked through the *string matching* of the whole statement using the Java program at the web server for online, or using the JavaScript program at the student browser for offline. The statement in the student answer and the corresponding statement in the original code are compared.

This marking process is executed at two levels in our implementation. The first level marking examines the statements after removing the spaces and tabs from the answers. To avoid confusions of novice students on use of spaces or tabs, this first level marking does not consider the spaces and tabs in the *string matching*.

However, to encourage a student to be aware of a readable code, the correct insertions of tabs and spaces are important in the code. Also, the indentation is of utmost importance in *Python programming* and plays a vital role in determining the syntax, structure, and readability of the code. Therefore, we need to carefully consider spaces, tabs, and indentations in program codes. As the result, the second level marking marks the statements including the spaces and tabs. If the answer statement contains a missing tab or space, or extra one, the warning message will be returned to the student in the answering interface.

4.5.1 First-Level Marking

First, the first-level marking applies to the answer. Here, after every space and tab is removed from the answer code, each statement is compared with the correct one without a space or tab. If they are different, the corresponding input form of the statement is highlighted by the pink background to suggest that at least one character in the answer statement is different from the correct one. Otherwise, the following second-level marking is applied.

4.5.2 Second-Level Marking

In the second-level marking, the whole statement, including the spaces and tabs, will be compared between the answer code and the original code. If they are different, it is highlighted by the yellow background. Otherwise, the form is not highlighted at all.

4.6 Summary

In this chapter, we reviewed the *Code Modification Problem (CMP)* and its generation procedure. Then, we presented the example *CMP* instance generation using a sample source code. Finally, we discussed two level answer marking for checking the answers in the *CMP* instance.

Chapter 5

Code Modification Problems for Data Visualization

This chapter presents the *code modification problem (CMP)* for data visualization in *Python Programming Learning Assistant System (PyPLAS)* [25].

5.1 Introduction

Data analysis or data science is one of the important applications in *Python programming*. Here, data visualization is necessary to graphically illustrate data using graphs or charts. In the *CMP* instance, one source code and two images showing graphs or charts are given. The first image represents the output one that is obtained by running the given source code. Then, it asks to modify the source code to output the second image. The correctness of the answer is checked through string matching with the correct one.

The rest of this chapter is organized as follows: Firstly, data visualization concepts are introduced. Secondly, the learning objectives of *CMP* instances related with data visualization are presented. Thirdly, the generation procedure of a *CMP* instance and the user interface are presented. Fourthly, the evaluation results through applications to students are presented. Finally, the conclusion of this chapter is presented.

5.2 Data Visualization in Python

In this section, we review the data visualization in *Python*.

5.2.1 Data Visualization Concepts

In today's world, a lot of data is being generated everyday. To analyze this data for certain trends, data visualization comes into play [27]. Data visualization is the graphical representation of information and data in pictorial or graphical formats such as charts, graphs, tables, and maps. It provides a good, organized pictorial representation of data and makes it easier to understand, observe, analyze. Data visualization tools support to see and understand trends, patterns in data, and outliers in data analysis. Therefore, data visualization tools and technologies are essential to analyzing massive amounts of information and making data-driven decisions.

Python has grown into a large community, further fueling the growth with new contributors and ecosystems. *Python* has many visualization tools/libraries to provide excellent features. They are easy to implement. They support all types of visual, live, customized charts. Most used *Python* libraries for data visualization are summarized as follows:

1. **Matplotlib:** It is a low-level library that provides much freedom to customize.
2. **Pandas Visualization:** Built on *Matplotlib*, it has an easy-to-use interface and makes visualization a breeze.
3. **Seaborn:** It has a high-level interface and many default styles.
4. **Bokeh:** It supports unique visualizations like network graphs, geospatial plots, etc.
5. **Plotly:** It can create interactive plots [26].

Data visualization in *Python* is essential for understanding data, communicating insights, making data-driven decisions, storytelling, exploration, analysis, and error detection. It enhances data understanding, facilitates effective communication, and empowers data-driven decision-making processes. In this thesis, we mainly focus on *Matplotlib* and *pandas* libraries for data visualization.

5.2.2 Types of Visualization Charts

By using the visualizations libraries, the following visualizations can be created [26]:

- **Line Chart**: A line chart is used to display trends over time. The X-axis usually represents a period, and the Y-axis represents the quantity associated with the period on the X-axis.
- **Area Chart**: An area chart is a line chart with the areas below the lines filled with colors. A stacked area chart displays each value's contribution to a total over some time.
- **Bar Chart**: A bar chart displays trends over time. In the case of multiple variables, a bar chart can make it easier to compare the data for each variable at every moment.
- **Histogram**: A histogram presents data using bars of different heights. Usually, each bar groups the numbers into ranges in a histogram. The taller the bars, the more data falls in that range. It displays the shape and the spread of continuous data set samples.
- **Scatter Plot**: A scatter plot is used to find correlations. If a data XY exists, a scatter plot is used to find the relationship between variables X and Y.
- **Bubble Chart**: A bubble chart is evolved from a scatter plot. Unlike a scatter plot, each data point is assigned a label or a category and is shown as a bubble. It is used to show and compare the relationship between the labeled circles. A bubble chart makes it hard to read with multiple bubbles, so it has a limited data set size capacity.
- **Pie Chart**: A pie chart is a circular graph representing the data set in which each pie slice represents a numeric proportion. A pie charts is used to show the contribution of a data point inside a whole data set.

- **Heat Map**: A heat map uses color like a bar chart's height and width. A heat map can identify whether the phenomenon is clustered or varies over space.

Figure 5.1 illustrates the above mentioned charts respectively.

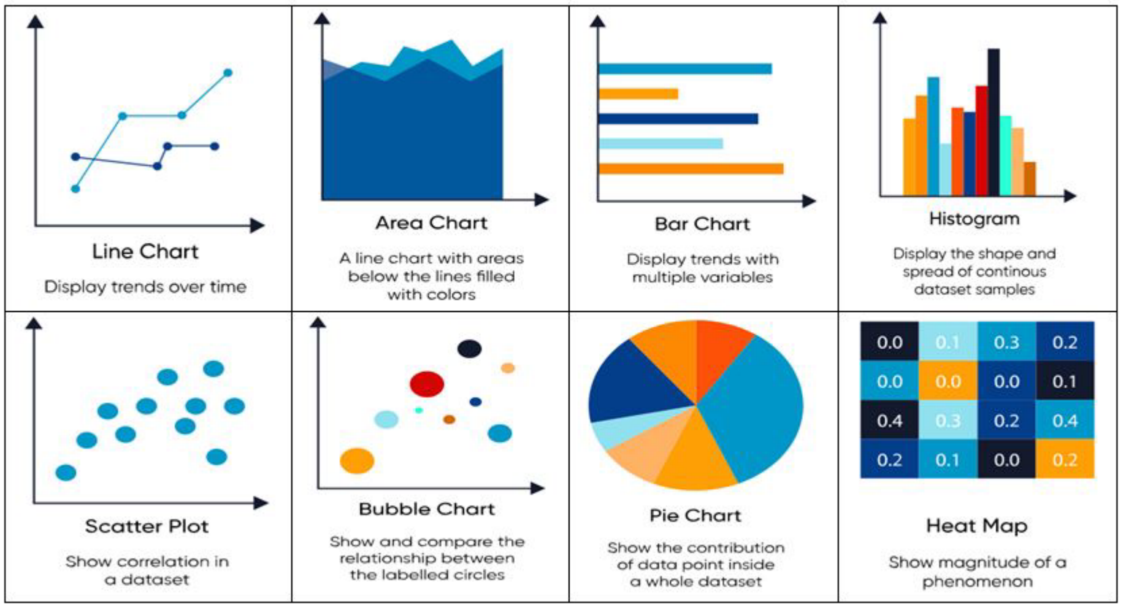

Line Chart
Display trends over time
Area Chart
A line chart with areas below the lines filled with colors
Bar Chart
Display trends with multiple variables
Histogram
Display the shape and spread of continous dataset samples
Scatter Plot
Show correlation in a dataset
Bubble Chart
Show and compare the relationship between the labelled circles
Pie Chart
Show the contribution of data point inside a whole dataset
0.0 0.1 0.3 0.2
0.0 0.0 0.0 0.1
0.4 0.3 0.2 0.4
0.2 0.1 0.0 0.2
Heat Map
Show magnitude of a phenomenon

Figure 5.1: Types of visualization charts.

5.3 Learning Objectives

To understand the benefits of data visualization, we generated 25 *CMP* instances with figures for important charts such as histogram, scatter plot, pie chart, venn diagram, and so on in Table 5.1.

5.4 CMP Instance Generation Procedure

A *CMP* instance can be generated through the following seven steps:

1. To select a source code from a website or a text book that is suitable for the current topic,
2. To find the important parts (functions, variables, parameters) in the source code to be modified so that the learner can study the instance topic,
3. To prepare another source code by replacing the modified parts from the original source code,
4. To put together the source code, the modified source code, and the correct answers into one text file,
5. To run the answer interface generation program with the text file as the input to generate the *CMP* instance with *HTML/CSS/JavaScript* files for the offline answering function,

6. To obtain the two images by running the original and modified source codes and add them into the generated *CMP* instance, and

7. To assign the generated *CMP* instance to students.

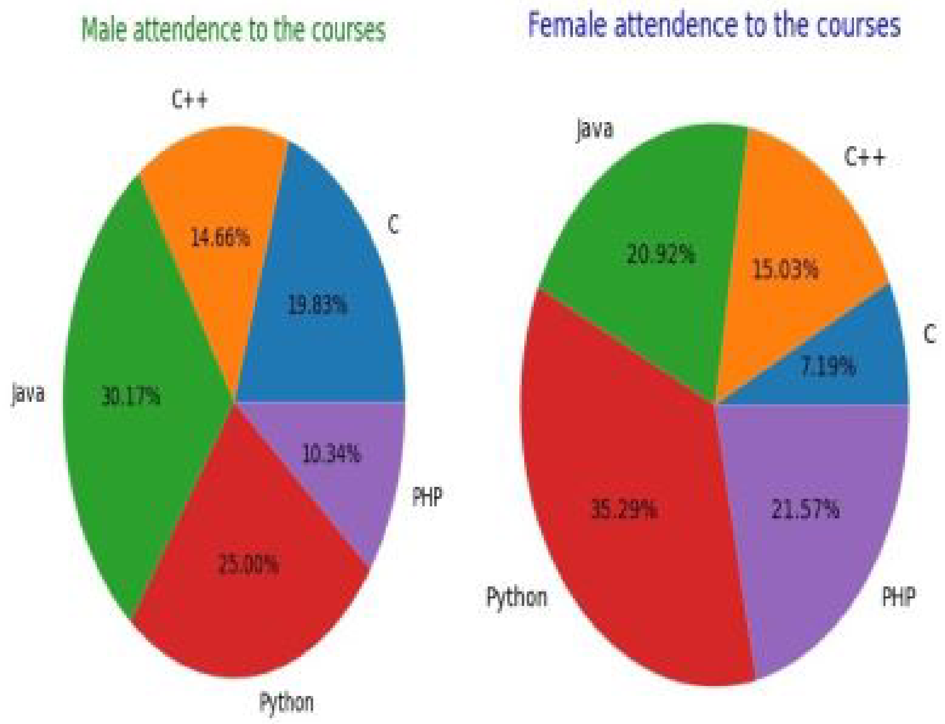
By using the Java programs and the Bash script, this procedure can be executed automatically.

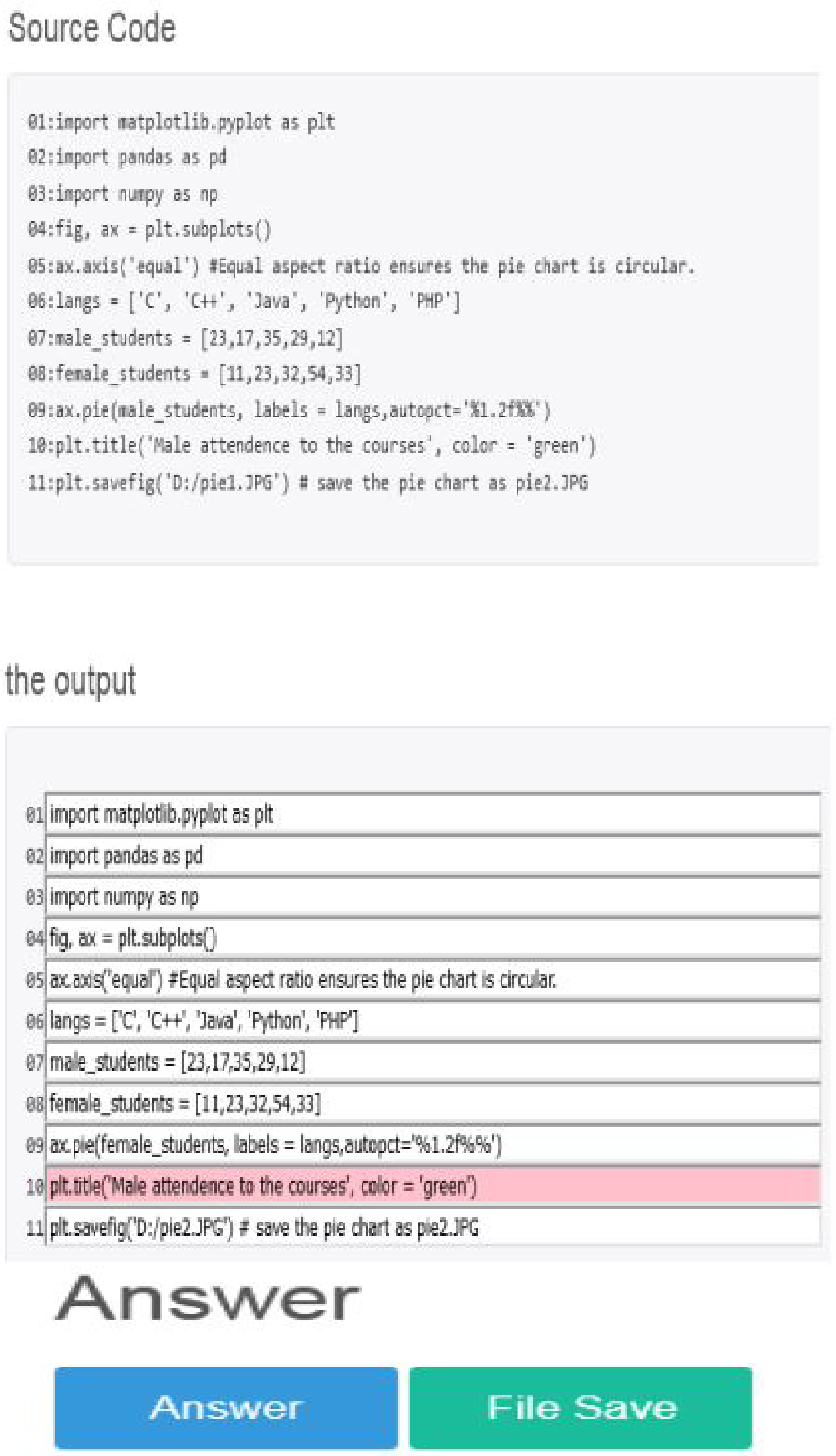
5.5 CMP User Interface

The answer interface for a *CMP* instance is implemented on the web browser. It allows a learner to solve the *CMP* instance both on online and offline, since the answer marking is processed by running the *JavaScript* program on the browser. The correct answers to the questions are encrypted using *SHA256* to avoid cheating by a student.

Figure 5.2 illustrates the answer interface for an example *CMP* instance. It asks three lines (line 9, line 10, and line 11) to be modified in the "Output" form. The first image was generated by running the given source code that illustrates the pie chart of male attendance on the respective courses. Then, to generate the right image, this problem asks the learner to properly modify the source code. Actually, it requests to change "male_students" in line 9 to "female_students", "Male attendance to the courses" and "green" to "Female attendance to the courses" and "blue" in line 10, and "pie1.JPG" to "pie2.JPG" in line 11. In this Figure, the learner still needs to modify line 10.

By understanding the given source code and the both images, the learner needs to modify the corresponding lines in the source code to obtain the second image by running the modified code. After the modification, the "Answer" button should be clicked. If the answer is not correct, the background color of the corresponding input form becomes pink. Otherwise, it is white. It is possible to submit answers repeatedly until all the answers become correct.




Source Code

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
fig, ax = plt.subplots()
ax.axis('equal') #Equal aspect ratio ensures the pie chart is circular.
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
male_students = [23,17,35,29,12]
female_students = [11,23,32,54,33]
ax.pie(male_students, labels = langs,autopct='%1.2f%%')
plt.title('Male attendence to the courses', color = 'green')
plt.savefig('D:/pie1.JPG') # save the pie chart as pie2.JPG
```

the output

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
fig, ax = plt.subplots()
ax.axis('equal') #Equal aspect ratio ensures the pie chart is circular.
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
male_students = [23,17,35,29,12]
female_students = [11,23,32,54,33]
ax.pie(female_students, labels = langs,autopct='%1.2f%%')
plt.title('Male attendence to the courses', color = 'green')
plt.savefig('D:/pie2.JPG') # save the pie chart as pie2.JPG
```

Figure 5.2: CMP user interface for data visualizations.

5.6 Evaluation

In this section, the evaluation of the proposal is discussed.

Table 5.1: Generated CMP instances for data visualizations.

Problem no	Problem name	Total blanks	Average Correct Rate
1	histogram	17	99%
2	scatter plot	20	100%
3	pie chart	11	100%
4	bar chart	16	99%
5	line graph	14	100%
6	stacked bar	17	98%
7	bubble chart	18	100%
8	lollipop	15	100%
9	waffle chart	9	98%
10	donut chart	13	98%
11	heatmap chart	24	99%
12	broken bar chart	15	98%
13	slopeplot chart	17	99%
14	area fill chart	12	100%
15	time series chart	20	100%
16	treemap chart	10	100%
17	venn diagram	9	100%
18	wordcloud chart	13	97%
19	multiple columns chart	12	100%
20	dumbbell chart	17	99%
21	sorted bar	11	98%
22	wedge pie chart	10	97%
23	stepplot chart	10	97%
24	combine several plots	14	100%
25	horizontal bar chart	12	100%
average		14.24	99%
total		356	SD: 1.1%

5.6.1 Discussion of generated CMP instances

For this evaluation, a total of 25 *CMP* instances are generated using source codes from the website[6] and the text book [7]. Table 5.1 shows the problem ID, the problem name, the total number of blanks and the average correct rate for each *CMP* instance by the students. As the topics, the bar chart, the line chart, the pie chart and others are selected. The validity of the *CMP* instances is evaluated through applications to 22 students in Okayama University who have studied *Python programming*.

Table 5.2: Correct answer rate distribution for data visualizations.

Correct Rate Range	# of Students
100%	13
99%	5
96%-98%	4

Table 5.3: Submission times distribution for data visualizations.

Submission Times Range	Avg.# of Submission(SD)	# of Students
25-65	40.8 (11.6)	8
66-106	98.7 (4.2)	5
107-147	112.5 (4.1)	4
148-188	167.7 (12.7)	3
189-229	207	1
230-270	265	1

5.6.2 Solution Results

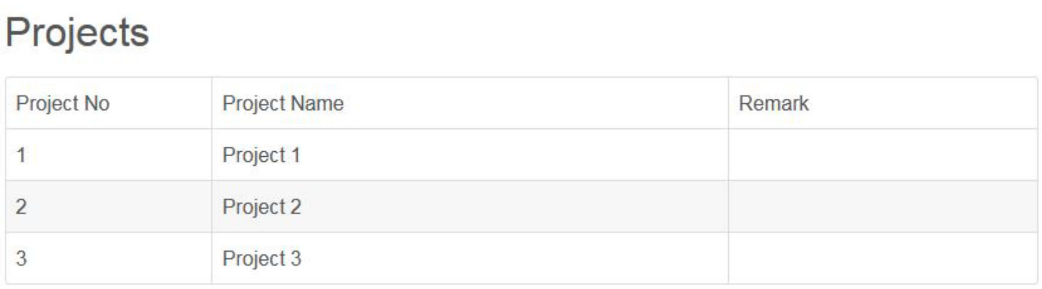
Table 5.1 shows that the average correct rate of the students is at least 97% for any instance, where the average correct rate is 99% and the average standard deviation SD is 1.1%. Thus, the generated *CMP* instances for data visualization are not difficult for these students.

Table 5.2 illustrates the distribution of the correct answer rates of the students. This table shows that 13 students among 22 correctly solved all the problems and five students achieved 99% correct rate. Only four students did it between 96% and 98%.

Table 5.3 shows the distribution of the number of answer submission times by the students. From this table, eight students submitted answers 1.6 times for each instance on average. It is noted that at least 25 submission times are necessary to solve all of the 25 instances. One student has this least number of submission times. One student spent 10.6 times on average for each instance. Therefore, these results clearly show that the generated *CMP* instances are generally proper to study the visualization concepts of *Python programming*.

5.7 Project Assignments

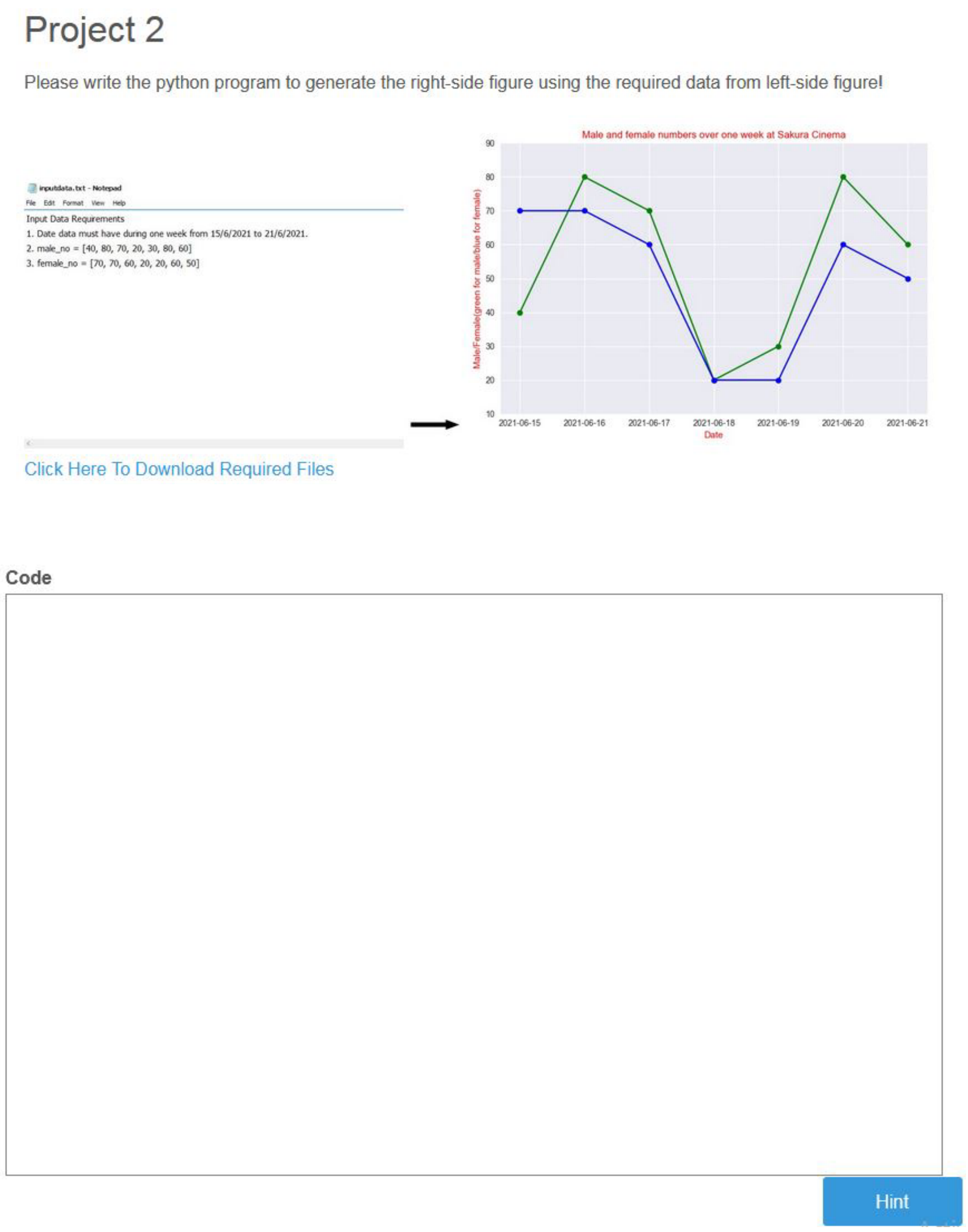
After assigning the generated *CMP* instances to the students, we implemented the project assignments to write the whole source code completely in the **"Code area"** that cover some of the generated *CMP* instances. The objective is to confirm whether or not students have actually understood the assigned *CMP* instances. The project lists and the user interface for an example project can be seen in Figure 5.3 and Figure 5.4. The hint function in Figure 5.5 is added in each project to assist the students. After finishing the each project assignment, the students can save their source codes by clicking "Save Code" button and send them to the teacher through the USB memories, or emails.



Projects

Project No	Project Name	Remark
1	Project 1	
2	Project 2	
3	Project 3	

Figure 5.3: Project lists.



Project 2

Please write the python program to generate the right-side figure using the required data from left-side figure!

inputdata.txt - Notepad

File Edit Format View Help

```
Input Data Requirements
1. Date data must have during one week from 15/6/2021 to 21/6/2021.
2. male_no = [40, 80, 70, 20, 30, 80, 60]
3. female_no = [70, 70, 60, 20, 20, 60, 50]
```

Male and female numbers over one week at Sakura Cinema
Male/Female(green for male/blue for female)
90
80
70
60
50
40
30
20
10
2021-06-15
2021-06-16
2021-06-17
2021-06-18
2021-06-19
2021-06-20
2021-06-21
Date

Click Here To Download Required Files

Code

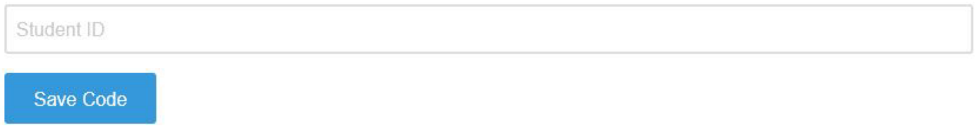
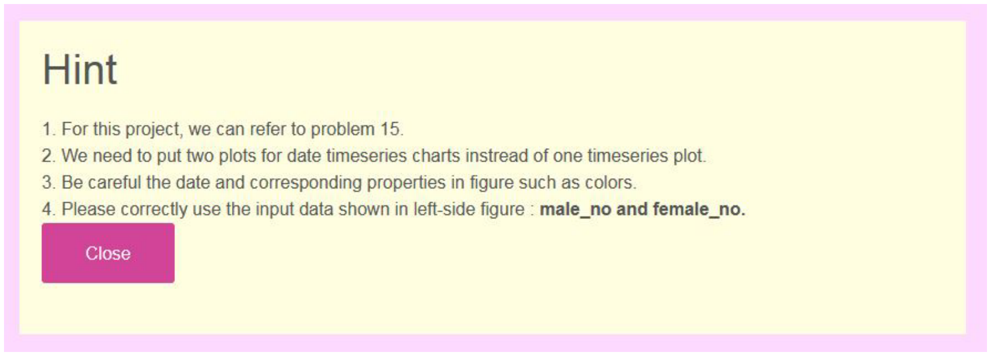


Figure 5.4: Project interface example.



Hint

1. For this project, we can refer to problem 15.
2. We need to put two plots for date timeseries charts instread of one timeseries plot.
3. Be careful the date and corresponding properties in figure such as colors.
4. Please correctly use the input data shown in left-side figure : **male_no and female_no.**

Figure 5.5: Hint function interface.

5.8 Summary

This chapter presented the *code modification problem (CMP)* for data visualization and the project assignments in *Python learning programming assistant system (PyPLAS)*. 25 *CMP* instances were generated using source codes in textbooks and websites that cover visualization such as line chart, bar chart, and so on. The application results to 22 students in Okayama University confirmed the effectiveness of the generated instances for self-study of *Python programming*. In future works, we will continue to generate *CMP* instances for other popular libraries such as *scikit-learn* and *spicy*.

Chapter 6

Code Modification Problems for Excel Problems

This chapter presents the code modification problem (CMP) for Excel Operations in *Python Programming Learning Assistant System (PyPLAS)* [28].

6.1 Introduction

In Chapter 5, we presented *Code Modification Problem (CMP)* for studying data visualization operations. In this chapter, we present *CMP* for *Excel* operations by extending the works. For data analysis, *Excel* is often used to manipulate a large amount of data and generate required graphs in short time. It has become entrenched in business processes worldwide for diverse functions and applications.

In this chapter, we focus on *CMP* for learning how to use *Python programming* libraries to manipulate data in *Excel* files. In a *CMP* instance for *Excel* operations in this chapter, generally, one source code and three images showing *Excel* files are given to the learner. The first image shows the input data file to the source code that will generate the output file from it. The output file is illustrated in the second image. The third image does another output file that will be generated by the answer source code that should be completed by the learner.

The answer code will be made by modifying the given source code after carefully checking the differences between the second and third images. By solving the *CMP* instances, the learner can master how to use *Python* libraries for common *Excel* operations. The correctness of the answer code is verified through string matching of each statement with the original code. A hint function is implemented for each *CMP* instance to assist learners in solving it.

We generated 25 *CMP* instances that cover *Python* libraries for important *Excel* operations through four steps. First, we collect the relevant source codes in the website and the text book. Second, we find and modify the important parts in each source code for understanding the *Excel* operations. Third, we run both the original and modified codes to obtain the outputs, and convert them into the image files. Finally, we generate the *HTML/CSS/JavaScript* files for this new instance that will be used in the *PyPLAS* answer function using a web browser.

The rest of this chapter is organized as follows: Firstly, the importance of *Excel* operations and learning objectives are introduced. Secondly, the *CMP* instance generation procedure, the user interface, and the generated *CMP* instances are presented. Thirdly, their evaluation results are described and analyzed. Finally, the conclusion of this chapter with future works is presented.

6.2 Importance of Excel Operations using Python

Excel is a common tool for data analysis across a range of fields, including banking, healthcare, and marketing, thanks to its versatility and usability. With *Excel*, the user can readily modify, examine, and display huge amounts of data, which makes it simpler to gain insights and make better choices. *Excel's* versatility lets users carry out a variety of data analysis activities from straightforward math operations to intricate statistical analysis.

However, it might often find repeating mundane tasks on a daily basis when working with *Excel*. These tasks may include copying and pasting data, formatting cells, and creating charts, among others. Over time, this can become monotonous and time-consuming, leaving the user with less time to focus on more important aspects of data analysis, such as identifying trends, outliers, and insights [29].

The drawbacks should be solved by automating *Excel* workflows with *Python*. The tasks like the spreadsheet consolidation, the data cleaning, and the predictive modeling can be done automatically using a simple *Python* script on *Excel* files. *Excel* users can also create a scheduler in *Python* that runs the script automatically at different time intervals, dramatically reducing the amount of human interventions required to perform the same tasks again [30].

Excel operations play significant roles in *Python* for various reasons. Here are some key points highlighting the importance of *Excel* operations in *Python*:

1. **Data Manipulation and Analysis**: *Python* allows reading, writing, and manipulating *Excel* files for data analysis tasks using libraries like *Pandas and NumPy*.

2. **Automation**: *Excel* is widely used for storing and managing data, but it can be time-consuming and error-prone to perform repetitive tasks manually. *Python* provides libraries like *pandas* and *openpyxl* to automate *Excel* operations.

3. **Integration with Other Libraries**: *Python* offers extensive libraries and frameworks for machine learning, data visualization, and scientific computing. *Excel* operations in *Python* can integrate with these libraries. For example, *Excel* data can be read into a *pandas* data frame, and train a machine learning model using *scikit-learn* or *TensorFlow*. The results will be back to *Excel* for reporting purposes.

4. **Excel as a Data Source**: Many organizations use *Excel* as data sources for various applications. With *Python*, data from *Excel* files can be extracted and integrated into software or database systems to streamline data pipelines and automate data extraction processes.

5. **Customization and Advanced Functionality**: The capabilities of *Excel* can be extended by using *Python* libraries to implement advanced mathematical operations, create complex formulas, generate charts, apply conditional formatting, and perform sophisticated data manipulations.

6. **Collaboration and Version Control**: By using *Python*, data can be extracted from *Excel*, and be processed independently. The results can be merged together, which enables smoother collaborations and version controls, allowing different team members to work on their parts of the analysis simultaneously.

The ability to perform *Excel* operations in *Python* increases the strengths of *Excel* and *Python*, and will provide a powerful combination for data manipulations, automations, integrations with other libraries, and customization options, making it an essential skill for data scientists, analysts, and developers working with *Excel* data in various domains. In this thesis, we mainly focus on *pandas* libraries for *Excel* operations with *Python*.

6.3 Python Pandas With Excel Sheet

pandas is the fast, powerful, flexible and easy library for open source data analysis and manipulations, built on top of the *Python*. It simplifies *Excel* operations in *Python* by providing a rich set of functionalities for data manipulations, analysis, integrations, data processing, visualizations, automations, and reproducibility, making it a valuable tool for working with *Excel* data in various domains. *pandas* is also useful for other data analysis tasks, such as:

- quick Exploratory Data Analysis (EDA)
- drawing attractive plots
- feeding data into machine learning tools like scikit-learn
- building machine learning models on the data
- taking cleaned and processed data to any number of data tools [31]

6.4 Learning Objectives

To study the benefits and practical use of *Excel* operations in *Python*, we generated 25 *CMP* instances using *pandas* library for important functions, such as merging one file to another file, creating pivot tables, searching value, replacing null value and so on in Table 6.1.

6.5 Generation of CMP instances for Excel Operations

A *CMP* instance can be generated through the following procedure:

1. To select one relevant *Python* source code from a website or a text book that contains the target library functions for this instance,
2. To prepare the input *Excel* file to the source code,
3. To find the important parts (functions, variables, or parameters) in the source code that should be modified by the learner to study this library,
4. To make the answer code by replacing the modified parts in the original source code,
5. To run the original and modified source codes to obtain the corresponding output *Excel* files,
6. To put together the source code, the modified source code, and the correct answers into one text file,
7. To run the program with this text file to generate the *HTML/CSS/JavaScript* files for the *PyPLAS* answer interface,

8. To modify the *HTML* file to add the images, and,
9. To assign the generated *CMP* instance to learners,

6.5.1 Source Code Selection

To explain the generation procedure of *CMP* for *Excel* operations, the following **source code** is used. This source code intends for learners to understand the sorting functions (ascending or descending order) of the required values via column names.

Listing 6.1: Source code example for excel CMP instance

```
1
2  import pandas as pd
3  import openpyxl
4  if __name__ == "__main__":
5      # read excel file
6      data = pd.read_excel('D:\Sortingvalues.xlsx')
7      df = pd.DataFrame(data)
8      # sorting the required values via column names and ascending or descending order
9      df=df.sort_values(by=['Code_no','Name'], ascending = True)
10     # saving to new excel file
11     df.to_excel('D:\Sortingvalues1.xlsx', index=True, header=True)
```

6.5.2 Preparation of Output Source Code

After selecting the source code, the teacher has to prepare another source code by finding and replacing the important parts in the original source code that should be focused on by the learner for the respective topic. **L**isting 6.2 illustrates the newly created source code with the modifications of line 9, and line 11 that are needed to be solved by the students.

Listing 6.2: Output code example for excel CMP instance

```
1
2  import pandas as pd
3  import openpyxl
4  if __name__ == "__main__":
5      # read excel file
6      data = pd.read_excel('D:\Sortingvalues.xlsx')
7      df = pd.DataFrame(data)
8      # sorting the required values via column names and ascending or descending order
9      df=df.sort_values(by=['No','Name'], ascending = False)
10     # saving to new excel file
11     df.to_excel('D:\Sortingvalues2.xlsx', index=True, header=True)
```

6.5.3 Generating Assignments

To generate the *CMP* instance with the *HTML/CSS/JavaScript* files for the answering interface, the teacher needs to put together the source code, the modified source code, and the correct answers into one text file with the required format for the *CMP* generator, and pass it as the input file to run the *CMP* generation program. After that, the teacher needs to add the required images to each generated HTML file so that the learner can carefully read the source code to know difference between the images for the correct answer. Then, the correct answers will be checked through string matching of two level answer marking.

6.5.4 Answer Interface

The answer interface for a *CMP* instance is implemented using a web browser. It allows a learner to solve the instance both on online and offline, since the answer marking is processed by running the JavaScript program on the web browser. Since the correct answers need to be distributed to the learners, they are encrypted using SHA256 to avoid cheating.

Figure 6.1 and Figure 6.2 illustrate the answer interface for the example *CMP* instance. In this instance, two lines (line 8 and line 10) in "Source Code" are requested to be modified in by a learner who needs to modify the corresponding lines in "Output".

The first image in Figure 6.1 represents the input *Excel* file for the given code and the modified code. The second image represents the output *Excel* file that is generated by running the given source code. It sorts the lines in the input file in ascending order of 'Code_no' where the tie is resolved by 'Name'. The third image does the output file that is obtained by sorting the lines in descending order of 'No' instead. Actually, it requests to change 'Code_no' and 'ascending = True' to 'No' and 'ascending = False' in line 8, and 'Sortingvalues1.xlsx' to 'Sortingvalues2.xlsx'in line 10. Figure 6.2 shows that the learner still needs to modify line 10.

After the learner modifies the source code in the input forms, he/she should click the "Answer". If the answer is not correct, the background color of the corresponding input form becomes pink. Otherwise, it is white. It is possible to submit answers repeatedly until all the answers become correct. Moreover, the 'Hint' button is implemented for helping the learner by explaining the running steps of the modified source code.

6.5.5 Generated CMP Instances

In this chapter, we generated 25 *CMP* instances using source codes in the website [32] and the text book [33]. Table 6.1 shows the instance number, the topic, the number of blanks for each *CMP* instance. Here, the average correct rate of the students in the next section is also included.

As the topics, we selected important *Excel* operations such as operations on multiple sheets, replacing missing values, sorting data values. These operations are essential in data analysis by cleaning, transforming, or reducing data.

6.6 Application Results

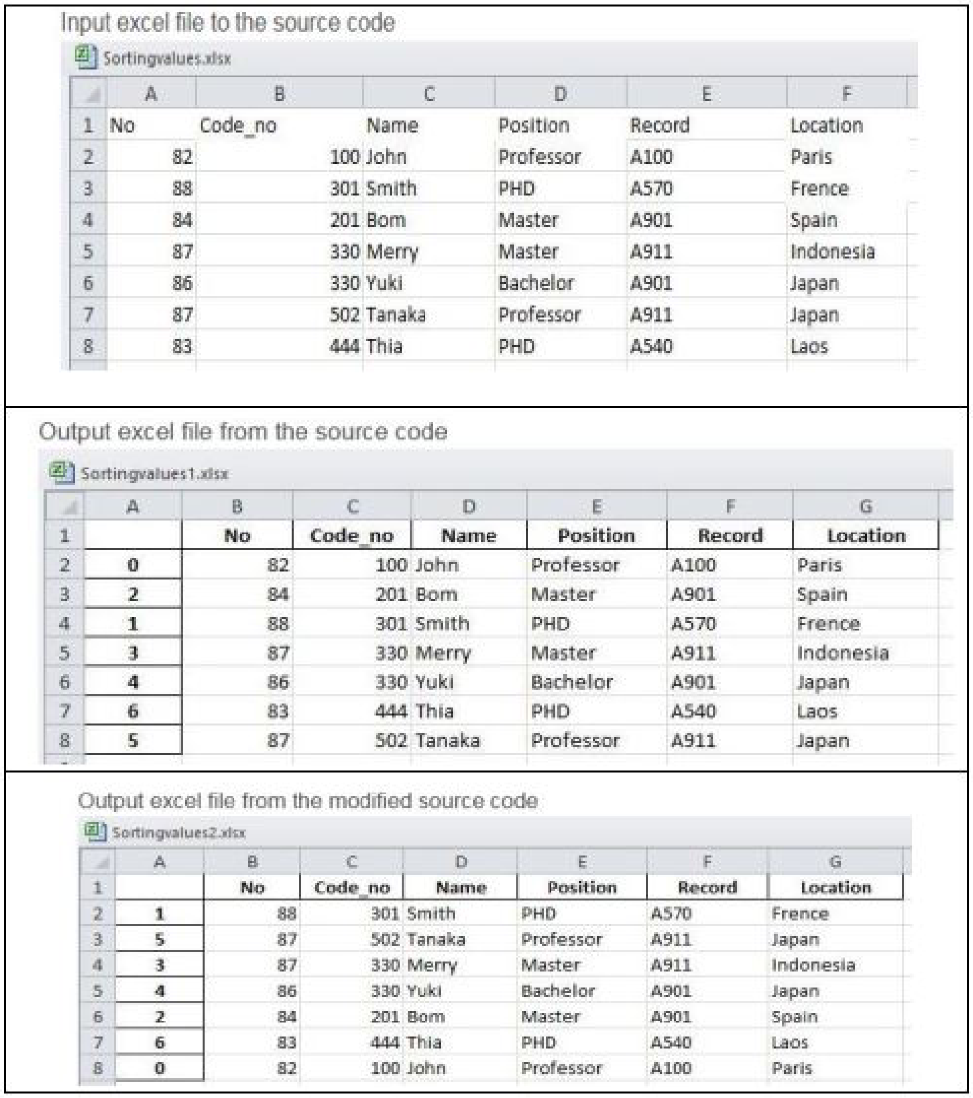
In this section, we evaluate the 25 *CMP* instances through applications to 13 undergraduate students in Okayama University, Japan, who have not studied *Python programming*.

6.6.1 Correct Rate for Each Instance

Table 6.1 indicates that the average correct rate among the students is 100% for the instances except for #1, #5, and #8, where the rate of the three instances is 99%. The overall average rate for all the instances is 99.89% and the standard deviation(SD) is 1.1%. Therefore, the generated *CMP* instances are not difficult for the students.

6.6.2 Correct Rate for Each Student

Table 6.2 shows the distribution of the correct answer rates of the students. It suggests that 10 students among 13 correctly solved all the *CMP* instances and three students achieved the 99%



Input excel file to the source code

Sortingvalues.xlsx

	A	B	C	D	E	F
1	No	Code_no	Name	Position	Record	Location
2	82	100	John	Professor	A100	Paris
3	88	301	Smith	PHD	A570	Frence
4	84	201	Bom	Master	A901	Spain
5	87	330	Merry	Master	A911	Indonesia
6	86	330	Yuki	Bachelor	A901	Japan
7	87	502	Tanaka	Professor	A911	Japan
8	83	444	Thia	PHD	A540	Laos

Output excel file from the source code

Sortingvalues1.xlsx

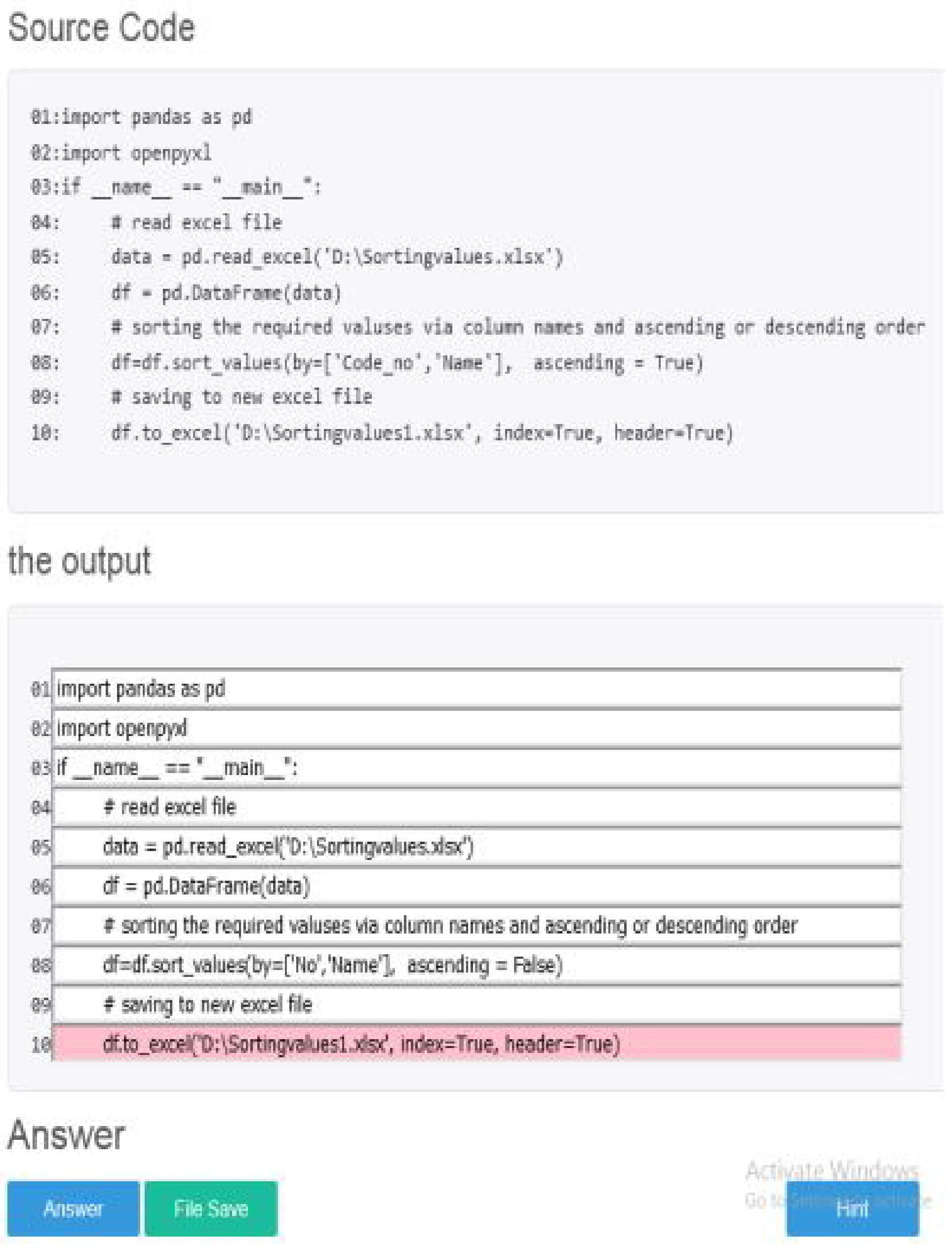
	A	B	C	D	E	F	G
1		No	Code_no	Name	Position	Record	Location
2	0	82	100	John	Professor	A100	Paris
3	2	84	201	Bom	Master	A901	Spain
4	1	88	301	Smith	PHD	A570	Frence
5	3	87	330	Merry	Master	A911	Indonesia
6	4	86	330	Yuki	Bachelor	A901	Japan
7	6	83	444	Thia	PHD	A540	Laos
8	5	87	502	Tanaka	Professor	A911	Japan

Output excel file from the modified source code

Sortingvalues2.xlsx

	A	B	C	D	E	F	G
1		No	Code_no	Name	Position	Record	Location
2	1	88	301	Smith	PHD	A570	Frence
3	5	87	502	Tanaka	Professor	A911	Japan
4	3	87	330	Merry	Master	A911	Indonesia
5	4	86	330	Yuki	Bachelor	A901	Japan
6	2	84	201	Bom	Master	A901	Spain
7	6	83	444	Thia	PHD	A540	Laos
8	0	82	100	John	Professor	A100	Paris

Figure 6.1: Three image for example CMP instance.



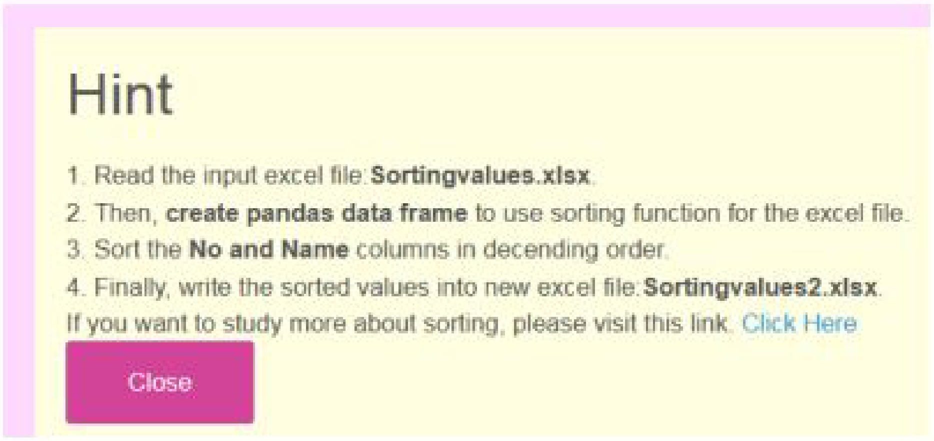
Source Code

```
import pandas as pd
import openpyxl
if __name__ == "__main__":
    # read excel file
    data = pd.read_excel('D:\Sortingvalues.xlsx')
    df = pd.DataFrame(data)
    # sorting the required valuses via column names and ascending or descending order
    df=df.sort_values(by=['Code_no','Name'],  ascending = True)
    # saving to new excel file
    df.to_excel('D:\Sortingvalues1.xlsx', index=True, header=True)
```

the output

```
import pandas as pd
import openpyxl
if __name__ == "__main__":
    # read excel file
    data = pd.read_excel('D:\Sortingvalues.xlsx')
    df = pd.DataFrame(data)
    # sorting the required valuses via column names and ascending or descending order
    df=df.sort_values(by=['No','Name'],  ascending = False)
    # saving to new excel file
    df.to_excel('D:\Sortingvalues1.xlsx', index=True, header=True)
```

Answer



Hint

1. Read the input excel file:**Sortingvalues.xlsx**.
2. Then, **create pandas data frame** to use sorting function for the excel file.
3. Sort the **No and Name** columns in decending order.
4. Finally, write the sorted values into new excel file:**Sortingvalues2.xlsx**.

If you want to study more about sorting, please visit this link. Click Here

Figure 6.2: Answer interface for example CMP instance.

correct rate.

6.6.3 Submission Times for each Student

Table **??** shows the distribution of the number of answer submission times by the students. This table suggests that for one instance, the minimum average number is 1.88 and the maximum one is 5.96. Therefore, the students can easily solve all the *CMP* instances. Thus, our proposal is generally proper to study *Python programming* libraries for *Excel* operations.

6.7 Project Assignments

After assigning the generated *CMP* instances, we implemented the project assignments to write the whole source code completely in the **"Code area"** that cover some of the generated *CMP* instances to confirm whether or not students have actually understood the assigned *CMP* instances. The user interface for an example project can be seen in Figure 6.3. The hint function is added in each project to assist the students. After finishing the each project assignment, the students can save their source codes by clicking "Save Code" button and send them to the teacher through the USB memories, or emails.

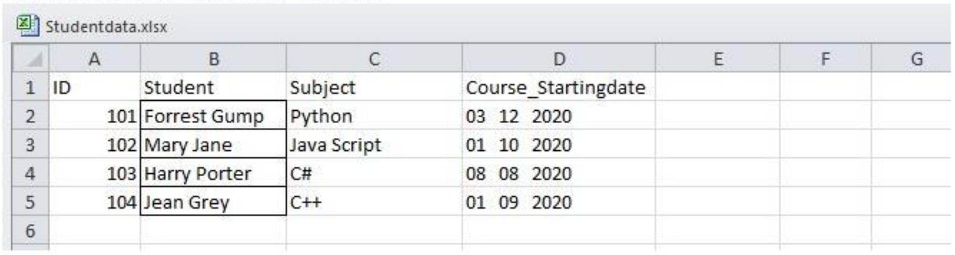
6.8 Summary

This chapter presented the *code modification problem (CMP)* for studying *Python programming* libraries on *Excel* operations in *Python Programming Learning Assistant System (PyPLAS)*. 25 *CMP* instances were generated using source codes in a textbook and a website that cover common *Excel* operations. The application results to 13 students in Okayama University confirmed the validity of the proposal with almost 100% correct rate for all instances and evaluated its effectiveness as a viable self-study tool. In future, we will generate *CMP* instances for other popular libraries for IoT (Internet of Things), machine learning, and multimedia.

Project 3

Please write the python program to generate the output excel file using the required data from the sheet1 and sheet2 of input excel file!

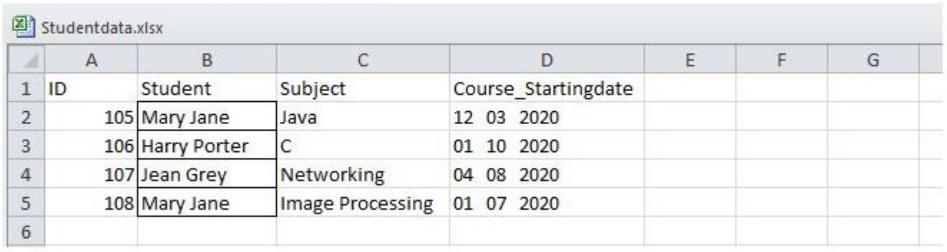
Input excel file (sheet_name=Sheet1)



Studentdata.xlsx

	A	B	C	D	E	F	G
1	ID	Student	Subject	Course_Startingdate			
2	101	Forrest Gump	Python	03 12 2020			
3	102	Mary Jane	Java Script	01 10 2020			
4	103	Harry Porter	C#	08 08 2020			
5	104	Jean Grey	C++	01 09 2020			
6							

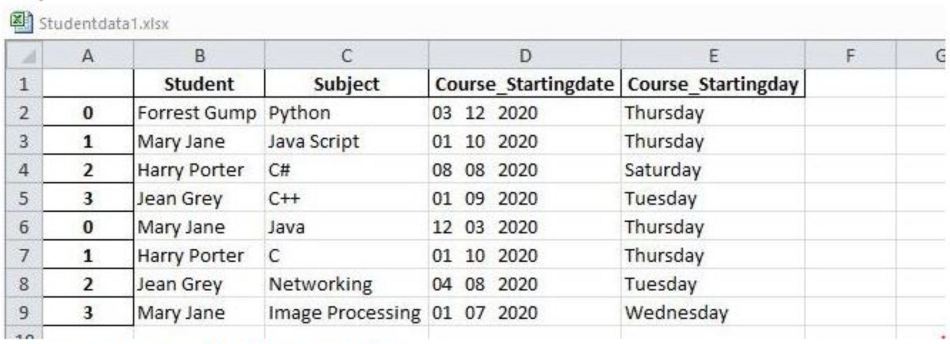
Input excel file (sheet_name=Sheet2)



Studentdata.xlsx

	A	B	C	D	E	F	G
1	ID	Student	Subject	Course_Startingdate			
2	105	Mary Jane	Java	12 03 2020			
3	106	Harry Porter	C	01 10 2020			
4	107	Jean Grey	Networking	04 08 2020			
5	108	Mary Jane	Image Processing	01 07 2020			
6							

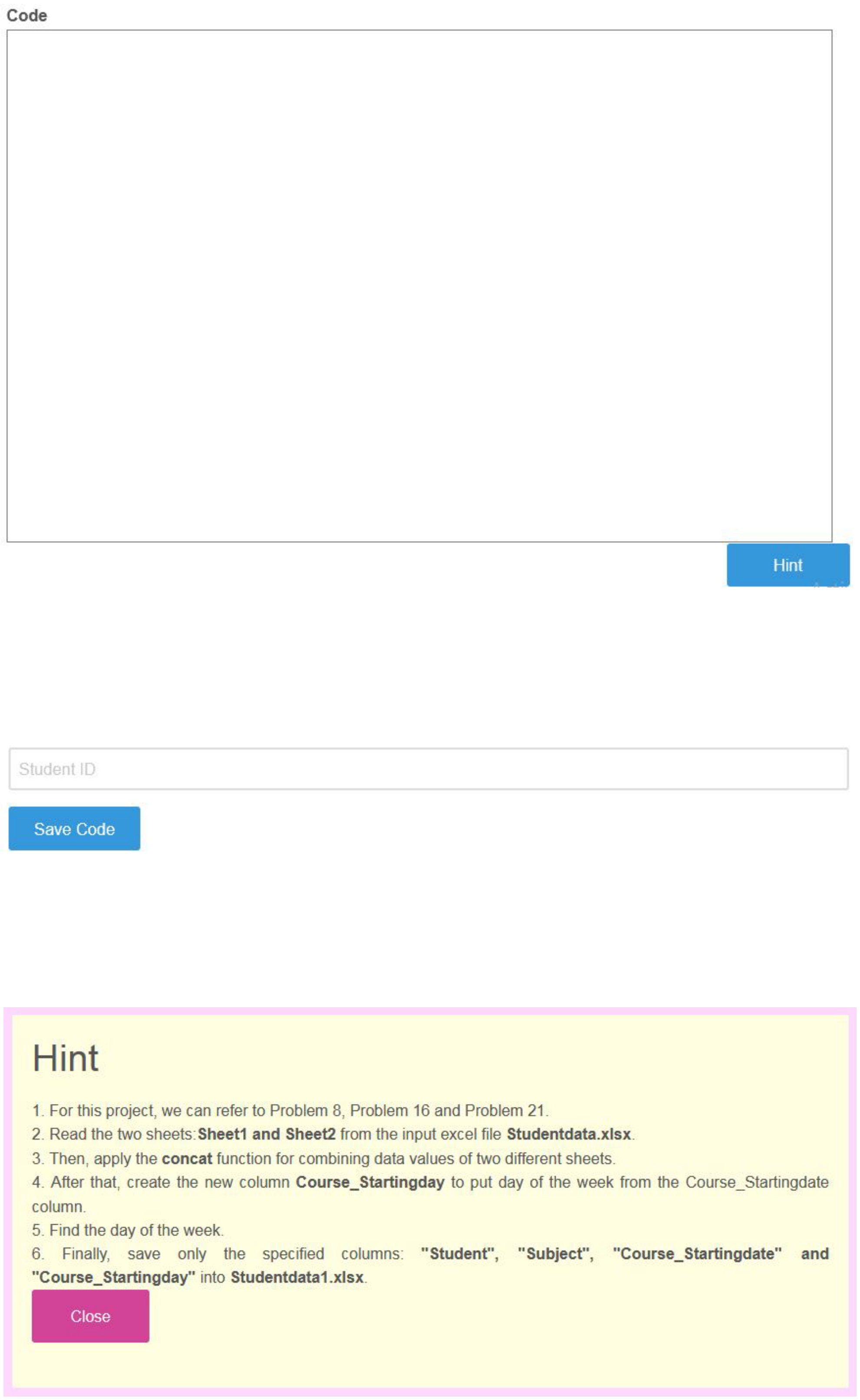
Output excel file



Studentdata1.xlsx

	A	B	C	D	E	F	G
1		**Student**	**Subject**	**Course_Startingdate**	**Course_Startingday**		
2	**0**	Forrest Gump	Python	03 12 2020	Thursday		
3	**1**	Mary Jane	Java Script	01 10 2020	Thursday		
4	**2**	Harry Porter	C#	08 08 2020	Saturday		
5	**3**	Jean Grey	C++	01 09 2020	Tuesday		
6	**0**	Mary Jane	Java	12 03 2020	Thursday		
7	**1**	Harry Porter	C	01 10 2020	Thursday		
8	**2**	Jean Grey	Networking	04 08 2020	Tuesday		
9	**3**	Mary Jane	Image Processing	01 07 2020	Wednesday		

Click Here To Download Required Files



Hint

1. For this project, we can refer to Problem 8, Problem 16 and Problem 21.
2. Read the two sheets: **Sheet1 and Sheet2** from the input excel file **Studentdata.xlsx**.
3. Then, apply the **concat** function for combining data values of two different sheets.
4. After that, create the new column **Course_Startingday** to put day of the week from the Course_Startingdate column.
5. Find the day of the week.
6. Finally, save only the specified columns: **"Student", "Subject", "Course_Startingdate" and "Course_Startingday"** into **Studentdata1.xlsx**.

Figure 6.3: Project interface example for Excel CMPs.

Table 6.1: Generated CMP instances for excel operations.

inst. #	topic	# of blanks	average correct rate
1	reading and writing excel files	5	99%
2	removing columns	4	100%
3	removing rows	5	100%
4	finding duplicate rows	5	100%
5	replacing null value	3	99%
6	calculating current age	3	100%
7	finding maximun and minimun date	4	100%
8	finding days of week	2	99%
9	operations on given date	3	100%
10	assigning new columns	4	100%
11	adding columns based on conditions	10	100%
12	renaming column names	9	100%
13	replacing multiple values	13	100%
14	sorting data values	11	100%
15	searching value	10	100%
16	separating files	10	100%
17	pivot table creation	7	100%
18	merge one file to another file	5	100%
19	merge on specific column	4	100%
20	concatenation of multiple sheets	5	100%
21	operations on multiple sheets	6	100%
22	writing data into csv file	4	100%
23	creating multiple csv files	5	100%
24	converting text file into csv file	5	100%
25	appending a new row to an existing csv file	6	100%
average		5.92	99.89%
total		148	(1.1%)

Table 6.2: Correct answer rate distribution for excel operations.

correct rate	# of students
100%	10
99%	3

Table 6.3: Submission times distribution for excel operations.

submission times range	average # of submissions (SD)	average # of submissions for one instance	# of students
25-50	47 (0)	1.88	1
51-76	69.5 (4.9)	2.78	2
77-102	97.5 (3.9)	3.9	4
103-128	113 (9.5)	4.52	3
129-154	149 (3.6)	5.96	3

Chapter 7

Related Works

In this section, we introduce related works to this thesis.

In [34], Garner presented learning resources and tools to aid novices learn programming. It is well known that the development of an introductory software is a difficult task for many students. The author discussed several resources and tools that are available or have been experimented with might be of interests to instructional designers of programming in the context of the four phases of the software lifecycle. The author analyzes the problem, designs a solution algorithm, implements the algorithm, and tests and revises the algorithm. The discussed tools include micro-worlds, video-clips, flowchart interpreters, and program animators.

In [35], Abu-Naser et al. presented an intelligent tutoring system for learning *Java* objects. By bringing together recent developments of tutoring systems, cognitive science, and artificial intelligence, they constructed an intelligent tutor system to help students learn *Java* programing.

In [36], Osman et al. introduced a visualized learning system to enhance the education of the data structure course. This system has the capability to display data structure graphically as well as allowing its graphical manipulations for a student to observe the execution result and to track the algorithm execution.

In [37], Li et al. presented a game-based learning environment to support novice students learning programming. It exploits game construction tasks to make the elementary programming more intuitive to learn, and comprises concept visualization techniques, to allow students to learn key concepts in programming through game object manipulation.

In [38], Hwang et al. proposed the web-based programming assisted system for designing learning activities for facilitating cooperative programming learning, and investigated cooperative programming learning behaviors of students and the relationships with learning performances.

In [39], Cai et al. studied performances of scientific applications with *Python* programming. They investigated several techniques for improving computational efficiencies of serial *Python* codes and discussed basic programming techniques for parallelizing serial scientific applications.

In [40], Bogdanchikov et al. suggested *Python* use for teaching programming to novice students, because the programming language has neatly organized syntax and powerful tools to solve any task. They gave some examples of program codes written in *Java*, *C++*, and *Python*, and made comparisons between them. They also pointed advantages of *Python*.

In [41], Adawadkar described the main features of *Python* programming and listed out the differences between *Python* and other programming language with helps of some codes. The author discussed applications of *Python* programming and showed good examples.

In [42], Alshaigy et al. presented a planned investigation on how learning styles and pedagogical methodologies can be embedded into an e-learning tool to assist learning programming.

Their main objective was to test the hypothesis that combining multiple teaching methods to accommodate different learners' preferences will significantly improve comprehensions of concepts, which in turn, increases students' confidences and performances in programming. An interactive learning tool to teach *Python* programming to students, called *PILeT*, had been developed to test the hypothesis. This tool is adaptable to the student's learning style and will teach programming using several techniques in visual, textual, and puzzles to appeal to each preference.

In [43], Zhang et al. provided new concepts and ideas for revealing human brain cognitive processes and promoting brain-inspired computing models. They used *Python* programming and its powerful technology ecosystem to provide supports for teaching and practicing the brain science as materials. They discussed how to use *Python* programming and corresponding development tools to complete the neuroimaging data preprocessing, the functional connectivity analysis, the multivoxel pattern analysis, and the searchlight analysis in the brain science teaching, where the corresponding practice processes were demonstrated with examples. From the results, the authors verified that students can better integrate with artificial intelligence technologies while learning brain science research techniques.

In [44], Brusilovsky et al. proposed the integrated practice system for learning programming in *Python* that provides an organized unified access to multiple types of smart practice contents.

In [45], Wang et al. designed programming exercises with computer assisted instructions by generating exercises at different levels of difficulty and sharing their experiences in design of teaching and learning activities for computer programming with large class size.

In [46], the author presented online *Python* tutor so that the teachers and students can write *Python* programs directly in the web browser (without installing any plugin), step forwards and backwards through executions to view the run-time state of data structure, and share their program visualizations on the web.

In [47], Helminen et al. introduced a program visualization and programming exercise tool for *Python* by aiming to target students apparent fragile knowledge of elementary programming, which manifests as difficulties in tracing and writing even simple programming. It provides an environment for visualizing the line-by-line execution of *Python* programs and for solving programming exercises with support for immediate automatic feedback and an integrated visual debugger.

In [48], the author applied a programming education tool called *pgtracer*, which they had developed as a Moodle plug-in, at an actual programming course to provide homework assignments to the students. They developed fill-in-the-blank questions based on the course syllabus at each week and evaluated the activities of students by using various functions provided by *pgtracer*. They also provided the analysis results to the teacher about the activities and achievements of the students for better collaborations between lectures and homeworks. They achieved the positive feedbacks by interviewing the teacher and surveying student's activities about the usefulness of *pgtracer*.

In [49], Rowe et al. implemented *VINCE* as an online tutorial tool for teaching introductory programming to allow the execution of a *C* program to be graphically displayed. The tool is written in *Java*, allowing it to be used on the web so that the student can enter their own *C* code, or select from a menu of pre-written tutorials, each illustrating a particular aspect of programming.

In [50], Sajaniemi et al. introduced a program animation system, *PlanAni*, based on the concept of the role of variables, which represents schematic uses of variables that occur in programs over and over, and a set of nine roles cover practically all variables in novice-level programs.

In [51], the author presented an automatic technique for generating maintainable regression unit tests for programs according to two main reasons compared to previous studies. First, they designed test generation techniques and evaluated upon libraries rather than applications. Second, they designed them to find bugs rather than to create maintainable regression test suites: the test suites that they generated were brittle and hard to understand. Thus, they presented a suite of techniques that address these problems by enhancing an existing unit test generation system. In experiments using an industrial system, the generated tests achieved good coverage and mutation kill scores, were readable by the product's developers, and required few edits as the system under test evolved.

In [52], Elenbogen et al. described a set of developed interactive web exercises and development environments designed to facilitate language acquisition in a beginning course in *C++*.

In [53], Stasko presented *TANGO*, a framework and system for algorithm animations by developing a conceptual framework with formal models and precise semantics for algorithm animations. The framework contains facilities for defining operations in an algorithm, designing animations, and mapping the algorithm operations to their corresponding animations.

In [54], Levy et al. developed the *Jeliot 2000* program animation system intending for teaching an introductory computer science to high school students with the goal of helping novices understand basic concepts of algorithms and programming such as assignment, I/O and control flow, whose dynamic aspects are not easily grasped just by looking at the static representation of an algorithm in a programming language.

Chapter 8

Conclusion

In this thesis, I presented the results of the three studies of exercise problems for *Python Programming Learning Assistant System (PyPLAS)*.

Firstly, I presented *the value trace problem (VTP)* for novice learners to study four groups of basic grammar, advanced grammar, data structure and algorithm, and library usage. For evaluations, I generated 130 source codes in *Python* programming textbooks or websites for the respective categories, and assigned them to 48 undergraduate students in Myanmar and Indonesia, and confirmed the validity of the proposal in *Python* programming self-studies by novice learners.

Secondly, I presented the *code modification problem (CMP)* for self-study of data visualizations in *Python* programming, as a new problem type for *PyPLAS*. For evaluations, 25 *CMP* instances were generated to cover important visualization concepts, and were assigned to 22 students in Okayama University, Japan. Their solution results show that the average correct rate for all of the 25 instances is at least 97%. Therefore, it confirmed that the difficulty levels of the *CMP* instances are proper for the novice learners as the viable self-study learning tools.

Thirdly, I presented the *code modification problem (CMP)* for studying how to use *Python* programming libraries to manipulate data in *Excel* file with the hint function implementation. For evaluations, I generated 25 *CMP* instances using *Python* codes for various *Excel* operations using *pandas* library and confirmed the validity of the proposal as a viable self-study tool with almost 100% correct rates for all instances to students in Okayama University. In future studies, I will further improve the self-study environments of *Python* by offering the remaining learning topics with various levels, including useful libraries (*scikit-learn*, *spicy*, etc.) for machine learning, deep learning, multimedia, IOT (Internet of Things), and so on for *VTP*, and *CMP* instances. Moreover, I will continue studying other types of problems with useful functions for each problem type for other *Python* programming topics in *PyPLAS*. Then, I will continue to assign the generated problems to students in programming courses.

Bibliography

[1] S. S. Wint, N. Funabiki, and M. Kuribayashi, “Design and implementation of desktop-version Java programming learning assistant system,” Proc. HISS, pp. 254-257, Nov. 2018.

[2] S. T. Aung, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N, K. Dim, and W.-C. Kao, ”A proposal of grammar- concept understanding problem in Java programming learning assistant system, “ to appear in J. Adv. Inform. Tech. (JAIT), Oct. 2021.

[3] K. K. Zaw, N. Funabiki, and W.-C. Kao, ”A proposal of value trace problem for algorithm code reading in Java programming learning assistant system,” Inform. Eng. Exp., vol. 1, no. 3, pp. 9-18, Sep. 2015

[4] N. Funabiki, H. Masaoka, N. Ishihara, I-W. Lai, and W.-C. Kao, ”Offline answering function for fill-in-blank problems in Java programming learning assistant system,” in Proc. ICCE-TW, pp. 324-325, May 2016.

[5] “Python Programs,” Internet: `https://www.geeksforgeeks.org/python-programming-examples/`, Access June 19, 2023.

[6] “Top 50 matplotlib Visualizations – The Master Plots (with full python code),” Internet:`https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/`, Access June 19, 2023.

[7] “Mastering Python Data Visualization,” Internet: `https://media.oiipdf.com/pdf/2447570f-0166-4f18-a2c0-15f7c3268605.pdf`, Access June 19, 2023.

[8] N. Funabiki, Y. Matsushim, T. Nakanishi, K. Watanabe, and N. Amano, “A Java programming learning assistant system using test-driven development method,” IAENG Int. J. Comput. Sci., vol. 40, no. 1, pp. 38-46, Feb. 2013.

[9] N. Ishihara, N. Funabiki, M. Kuribayashi, and W.-C. Kao, “A software architecture for Java programming learning assistant system,” J. Comp. Soft. Eng., vol. 2, no. 1, Sept. 2017.

[10] N. Ishihara, N. Funabiki, and W.-C. Kao, “A proposal of statement fill-in-blank problem using program dependence graph in Java programming learning assistant system,” Info. Engr. Exp., vol. 1, no. 3, pp. 19-28, Sept. 2015.

[11] N. Funabiki, Tana, K.K. Zaw, N. Ishihara, and W.-C. Kao, ”A graph- based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system. IAENG Int J Computer Science 44: 2.

[12] K. Beck, Test-driven development: by example, Addison-Wesley,2002.

[13] H.H.S. Kyaw, N. Funabiki, and W.-C. Kao, “A proposal of code amendment problem in Java programming learning assistant system,” International Journal of Information and Education Technology (IJIET), vol. 10, No. 10, pp. 751-756, Oct. 2020.

[14] H.H.S. Kyaw, S.S. Wint, N. Funabiki, and W.-C. Kao, “A code completion problem in Java programming learning assistant system,” IAENG International Journal of Computer Science (IJCS), vol. 47, No. 3, pp. 350-359, Sept. 2020.

[15] “SHA-256 Cryptographic Hash Algorithm,” Internet: `http://www.movable\protect\discretionary{\char\hyphenchar\font}{}{}type.co.uk/scripts/sha256.html/`, Access June 20, 2023.

[16] S. T. Aung, N. Funabiki, L. H. Aung, H. Htet, H. H. S. Kyaw, and S. Sugawara, ” An Implementation of Java Programming Learning Assistant System Platform Using Node.js,” ICIET International Conference of Information and Education Technology (ICIET), pp. 47-52, Apr.2022.

[17] D. Herron, Node.js web development, Birmingham, UK, 2016.

[18] “Express,” Internet: `https://expressjs.com/.`, Access June 20, 2023.

[19] R. McKendrick, Monitoring Docker, United Kingdom, 2015.

[20] A. Mouat, Using Docker: developing and deploying software with containers, USA, 2015.

[21] “JUnit 5,” Internet: `https://junit.org/junit5/`, Access June 20, 2023. .

[22] “What is Python The most versatile programming language,” Internet: `https://www.datacamp.com/blog/all-about-python-the-most-versatile-programming-language/`, Access June 20, 2023.

[23] S. H. M. Shwe, N. Funabiki, Y. W. Syaifudin, E. E. Htet, H. H. S. Kyaw, P. P. Tar, N. W. Min, T. Myint, H. A. Thant, and W.-C. Kao, “Value trace problems with assisting references for Python programming selfstudy,” International Journal of Web Information Systems, Jun. 2021.

[24] “A Byte of Python,” Internet: `https://www.ibiblio.org/swaroopch/byteofpython/files/120/byteofpython_120.pdf`, Access June 19, 2023.

[25] S. H. M. Shwe, N. Funabiki, H. H. S. Kyaw, K. H. Wai and W.-C. Kao, “A proposal of code modification problem for Python programming learning assistant system,” International Symposium on Socially and Technically Symbiotic Systems (STSS), November 15-17, 2021.

[26] “Importance And Benefits of Data Visualization,” Internet: `https://www.mindbowser.com/benefits-of-data-visualization/`, Access June 19, 2023.

[27] “Data Visualization with Python,” Internet: `https://www.geeksforgeeks.org/data-visualization-with-python/`, Access June 19, 2023.

[28] S. H. M. Shwe, N. Funabiki, K. H. Wai, S. L. Aung, and W.-C. Kao, “A study of code modification problems for Excel operations in Python programming learning assistant system,” 2022 10th International Conference on information and Education Technology (ICIET 2022), pp. 209-213, April 9-11, 2022.

[29] “How to automate Excel tasks with Python,” Internet: `https://www.freecodecamp.org/news/automate-excel-tasks-with-python/`, Access June 19, 2023.

[30] “Python Excel tutorial: the definitive guide,” Internet: `https://www.datacamp.com/tutorial/python-excel-tutorial/`, Access June 19, 2023.

[31] “Tutorial using Excel with Python and Pandas,” Internet: `https://www.dataquest.io/blog/excel-and-pandas/`, Access June 19, 2023.

[32] “Working with excel files using Pandas,” Internet: `https://www.geeksforgeeks.org/working-with-excel-files-using-pandas/`, Access June 19, 2023.

[33] “Working with Excel spreadsheets,” Internet: `https://automatetheboringstuff.com/chapter12.pdf`, Access June 19, 2023.

[34] S. S. Garner, ”Learning resources and tools to aid novices learn programming,” in Proc. Int. Conf. Informing Science, vol. 2, no. 2, June 2003.

[35] S. Abu-Naser, A. Ahmed, N. Al-Masri, A. Deeb, E. Moshtaha, and M. Abu-Lamdy, ”An intelligent tutoring system for learning Java objects,” Int. J. Art. Intell. Appli., vol. 2, no. 2, pp. 68-77, April 2011.

[36] W. I. Osman and M. M. Elmusharaf, ”Effectiveness of combining algorithm and program animation: a case study with data structures courses,” Issue. Inform. Sci. Inform. Tech., vol. 11, pp. 155-168, 2014.

[37] F. W. B. Li and C. Watson, ”Game-based concept visualization for learning programming,” in Proc. MTDL, pp. 37-42, Dec. 2011.

[38] W.-Y. Hwang, R. Shadiev, C.-Y. Wang, and Z.-H. Huang, ”A pilot study of cooperative programming learning behavior and its relationship with students’ learning performance,” Comput. Edu. vol. 58, pp. 1267–1281, 2012.

[39] X. Cai and H. P. Langtangen, ”On the performance of the Python programming language for serial and parallel scientific computations,” Sci. Programm., vol. 13, no. 1, pp. 31-56, Jan. 2005.

[40] A. Bogdanchikov, M. Zhaparov, and R. Suliyev, ”Python to learn programming,” J. Physics: Conf. Series, vol. 432, 2013.

[41] K. Adawadkar, ”Python programming - applications and future,” Sci. J. Impact Factor, April 2017.

[42] B. Alshaigy, B. Alshaigy, S. Kamal, F. Mitchell, C. Martin, and A.Aldea, ”PILeT: an interactive learning tool to teach Python,” in Proc. WiPSCE, pp. 76-79, Nov. 2015.

[43] X. Zhang, J. Huang, Y. Yang, X. He, R. Liu, and N. Zhong, ”Applying Python in brain science education,” in Proc. IJCIME, Dec. 2019.

[44] P. Brusilovsky, L. Malmi, R. Hosseini, J. Guerra, T.Sirkia, and K.Pollari-Malmi, ”An integrated practice system for learning programming in Python: design and evaluation,” RPTEL vol. 13, no. 18, 2018.

[45] F. L. Wang and T. Wong, “Designing programming exercises with computer assisted instruction,” in Proc. Int. Conf. ICHL, Aug 2008.

[46] P. J. Guo, ”Online Python tutor: embeddable web-based program visualization for CS education,” in Proc. ACM Tech. Symp. Comput. Sci. Edu., pp. 579-584, Mar. 2013.

[47] J. Helminen and L. Malmi ”JYPE- a program visualization and programming exercise tool for Python,” in Proc. Int. Symp. Soft. Visual., pp 153-162, Oct. 2010.

[48] T. Kakeshita and M. Murata, ”Application of Programming Education Support Tool pgtracer for Homework Assignment,” J. Learn. Tech. Learn. Envr, vol. 1. no.1, pp. 41-60, Mar. 2018.

[49] G. Rowe, and G. Thorburn, “VINCE - an on-line tutorial tool for teaching introductory programming,” in Proc. ACM SIGCSE Bulletin, vol. 30, no. 3, Sept. 1998.

[50] J. Sajaniemi, and M. Kuittinen, ” Program animation based on the roles of variables,” in Proc. ACM Symp. Soft. Visual., pp. 7-16, Jun. 2003.

[51] B. Robinson, M. D. Ernst, J. H. Perkins, V. Augustine and N. Li, ”Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs,” 26th IEEE/ACM Int. Conf. on Auto. SW Eng. (ASE 2011), pp. 23-32, 2011.

[52] B. S. Elenbogen, B. R. Maxim, and C. McDonald, ”Yet, more web exercises for learning C++,” in Proc. ACM SIGCSE Bulletin, vol. 32, no.1, pp. 290-294, May. 2000.

[53] J. T. Stasko, “Tango: a framework and system for algorithm animation,” IEEE Computers, vol. 23, no. 9, pp. 27-39, Sept. 1990.

[54] R.B.Levy, M.Ben-Ari, and P.A.Uronen, “The jeliot 2000 program animation system”, Comput. Edu., vol. 40, no. 1, pp. 1-15, Jan. 2003.