Tutor Bot: Tutoring System

Group Members: Jenny Goldsher, Noah Harvey, Deandra Martin, Hiroki Sato

Course: Software Engineering

Professor: Dr. Elva

Date: December 16, 2020

Table of Contents

SDLC Phases:

Planning

Students in CMS 120 have come to tutoring sessions with several consistent issues. For instance, students write algorithms with incorrect data types for the variables they use. This type of issue comes from how Python is formatted (not making programmers declare data types on assignment/declaration) and obfuscates details from programmers. Although the format of Python is oriented towards making programming easier, beginners do not inherently learn necessary concepts because the language covers up many aspects of programming complexity for the sake of ease of use.

Because CMS 120 is an introductory course, students are not expected to have prior knowledge of these concepts, however, it is good practice to teach students to code defensively and understand how the code they have written works. Since CMS 120 students are beginners coding in python – which allows them to program with more freedom than many other strongly typed languages – they may face more difficulty when they reach upper level CS classes if they never have a chance to develop a proper understanding of key programming concepts. This emphasizes the importance of having beginner students learn effective coding practices early on.

To combat this problem, there is a need for a web application that can read through student code snippets, give feedback, and ask questions. The program will not format the code for students, but it will ask related questions and challenge them to further their understanding of python. This would be a comprehensive way for CMS 120 students to exercise and reflect on their understanding of good defensive coding practice taught in class while they are programming such as being aware of data types of variables, reducing the number of unnecessary declaration and assignment of variables.

Requirements Engineering

Requirements Elicitation

To formulate the requirements, we gathered information from CS tutors: Jenny and Hiro by facilitating two JAD Sessions. We scheduled meetings and reviewed other related applications (Slack, CodingBat, LeetCode). The tutors first described the project generally independent of a specific platform which allowed us to establish a mutual and solid understanding of what we wanted to achieve.

In the meeting we discussed topics such as:

- The objective of this project/what the client is expecting to get out of this project
- How students should be able to interact with the system
- The types of questions the system should generate
- How the system should respond to student's input

By holding a JAD session, we were able to pinpoint contradictions, ambiguities, and misinterpretation that could have occurred between clients and developers in real time. Further, we were able to discuss and address those issues. In addition, having both clients and developers in a meeting allowed us to make decisions that both sides could agree on quickly. We discussed whether to allow the system to format the code they received, and we decided that it would not – in keeping with the honor code of Rollins College to avoid unauthorized assistance. We also clarified that the system would not execute code passed by the student as this represents a security vulnerability and depending on its use could also violate Rollin's Honor Code. We further discussed general types of questions the system would challenge the student with and clarified that they cannot be open-ended.

Requirements Analysis

We reviewed the information we had gathered and polished our understanding of what the system needs to be able to do as well as how it should execute these tasks after the JAD session. During our review and discussion within the development team, we realized that students should be able to feed in only a snippet of code instead of the entire Python file. This was the most logical strategy as the file was not getting compiled but parsed, and it would make students practice their coding in real time. Uploading entire files could also be a security vulnerability where the integrity of the system might be compromised by flawed files.

We also briefly discussed what software development model we should use for this project and agreed to use an incremental model. We chose this model based on its increased flexibility that would give us liberty to easily add or remove features. With short development time and a novel environment, this increased flexibility would be important.

Requirements Specification

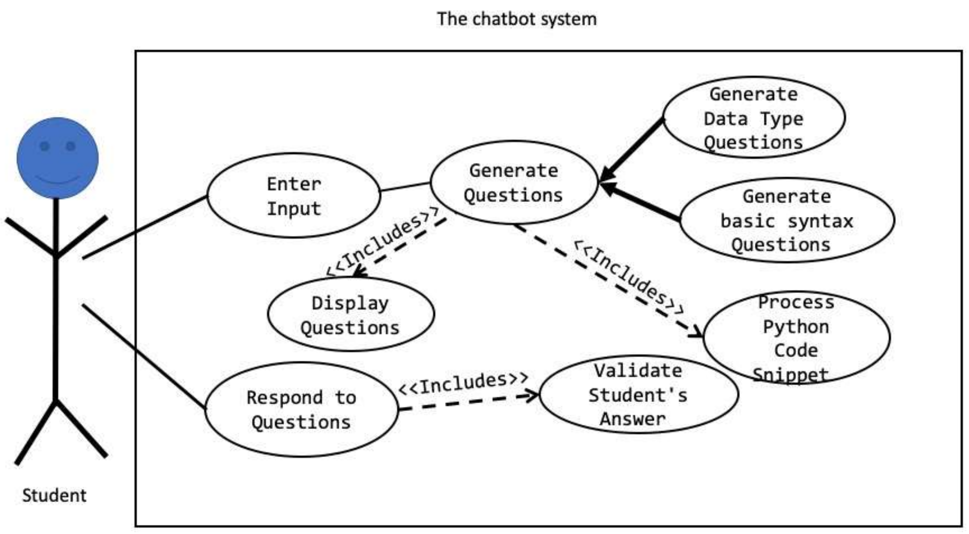
After meeting with CMS tutors and conducting our analysis, we came up with a list of requirements:

- Functional Requirements:
    - The system should allow users to enter a snippet of Python code.
    - The system should generate a set of questions based on the Python code that the user feeds in.
    - The system is a parser, not a compiler. This program should not execute the Python file and provide its output.
    - The system should provide a link to a page that discusses common error messages and their meaning.

    - The system should generate questions such as:
        - Data type questions
        - Basic syntax questions
        - Questions about functions
- Non-functional requirements
    - The system should be accessible to all students via the internet (Web application)
    - It should be clear to user where they should insert their code snippet
    - The application should be in the form of a chatbot where user can easily interact
    - The system should generate answers but not show the answer until students at least try to answer those questions on their own first.
    - The system does not necessarily have to display all the possible questions, the number of questions can be selected or randomly generated (Default is 7).
    - The system should be implemented on a Linux server and using Python 3.

Design

Use Case Diagram

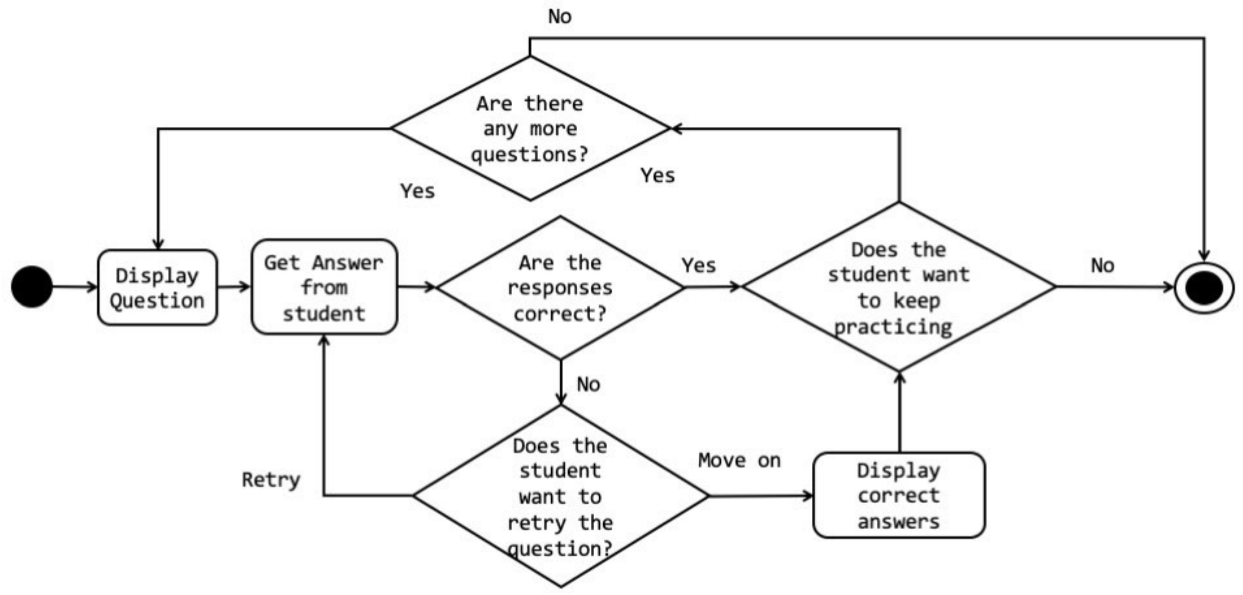


We also modelled our understanding of how students should be able to interact with the system using a Use Case diagram. Users will interact with the system when they insert the snippet of code or when they type in their answers to the questions generated by the system.

In addition to that, we agreed that we should use a combination of incremental approach and integration and configuration approach.

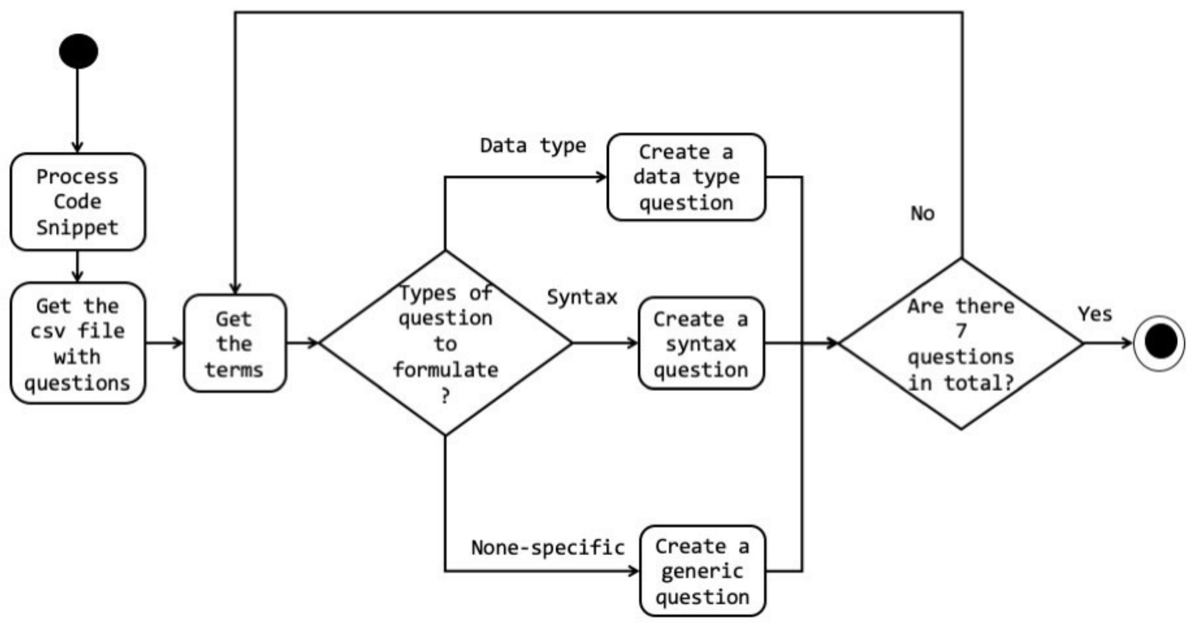
Activity Diagrams

Activity Diagram for Validate Student's Input


No
Are there any more questions?
Yes
Yes
Display Question
Get Answer from student
Are the responses correct?
Yes
Does the student want to keep practicing
No
No
Retry
Does the student want to retry the question?
Move on
Display correct answers

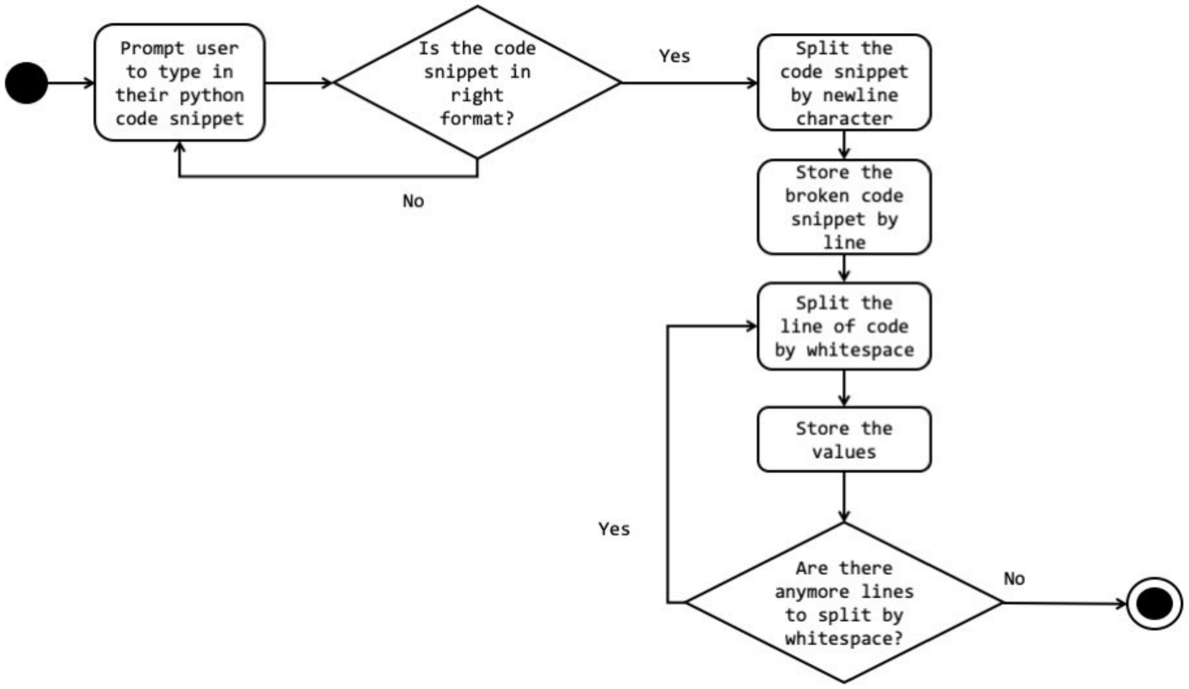
The tutor bot should display the default 7 random questions and prompt the student to respond. If their responses are correct, the student is given the option to keep practicing - which would display 7 more questions if available - or not. However, if at least one of the responses is incorrect, the student is given the option to retry and re-enter the correct answer or move on and display the correct answers. In this case, the student is free to keep practicing or not.

Activity Diagram for Generate Questions


Process Code Snippet
Get the csv file with questions
Get the terms
Types of question to formulate ?
Data type
Syntax
None-specific
Create a data type question
Create a syntax question
Create a generic question
No
Are there 7 questions in total?
Yes

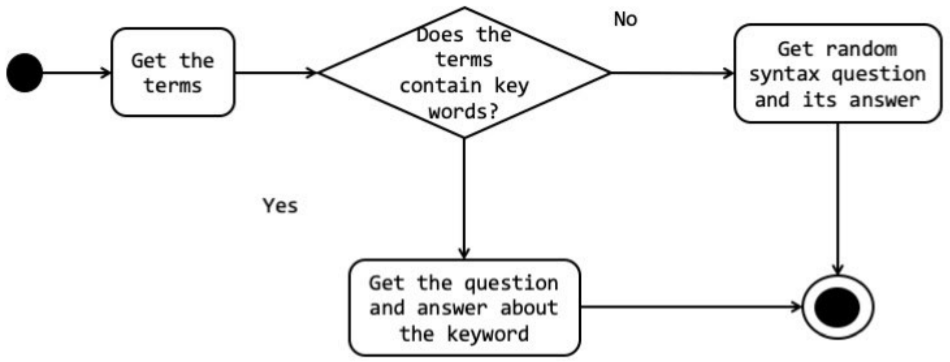
The system is going to process the code snippet created by the user and split it into tokens. We also get the .csv file with questions and based on the tokens we are going to create questions about either data type, python programming syntax or generic questions. Generic questions are not specific to syntax or data type but are about python or programming in general. This route will be taken by the system when we do not have enough code snippets provided by the user.

Activity Diagram for Process Code Snippet


Prompt user to type in their python code snippet
Is the code snippet in right format?
Yes
No
Split the code snippet by newline character
Store the broken code snippet by line
Split the line of code by whitespace
Store the values
Yes
Are there anymore lines to split by whitespace?
No

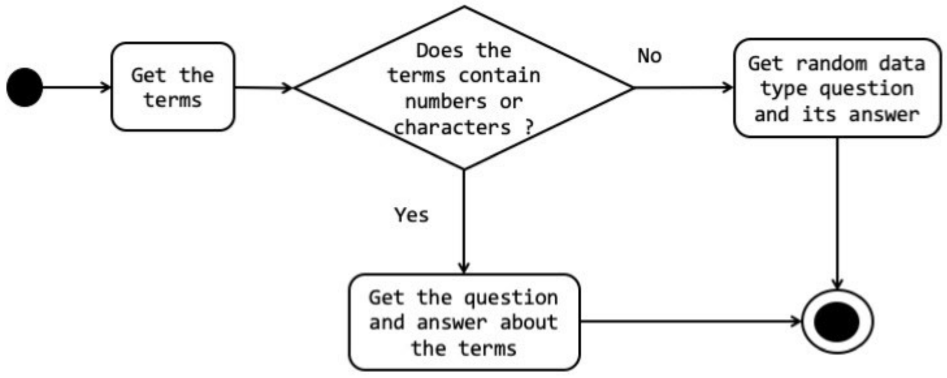
The bot will prompt students to enter their line of code in Python. The bot should determine if the snippet is in the right format - if it is, the code is split by a new line character and the broken code is stored by line. The line of code is then split by whitespace and stored as values. If there are more lines, the code is split and stored until there are no more lines to process. Should the format of the entered code snippet be incorrect, the bot will prompt the user again to re-enter their snippet.

Generate Basic Syntax Question


Get the terms
Does the terms contain key words?
No
Get random syntax question and its answer
Yes
Get the question and answer about the keyword

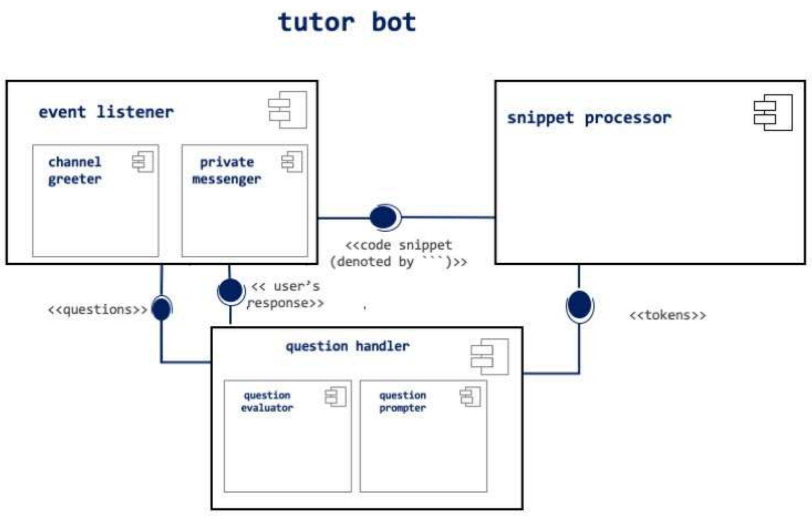
If the code snippet provided contained tokens that matched the keywords for syntax type questions, then it should generate a question about the keyword. Otherwise, it should generate a random syntax question.

Generate Data type question


Get the terms
Does the terms contain numbers or characters ?
No
Get random data type question and its answer
Yes
Get the question and answer about the terms

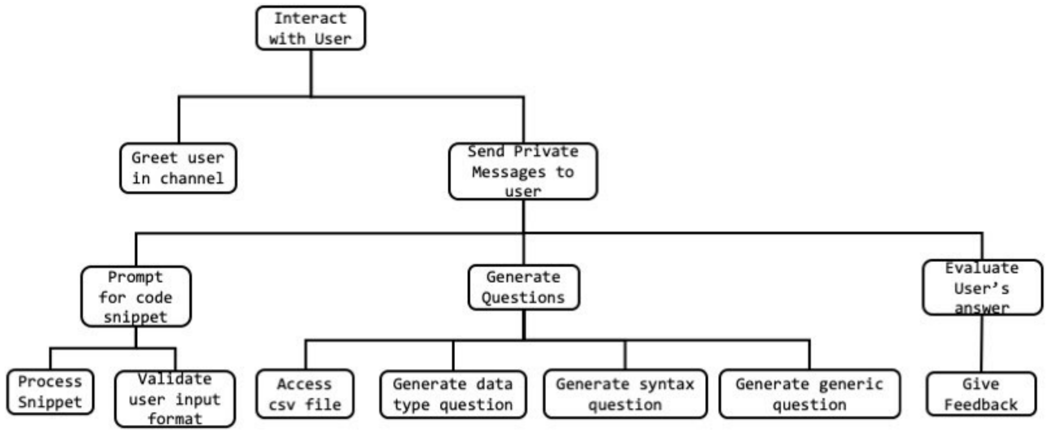
If the code snippets provided were related to data types such as variable declaration with value, and functions with return statements with values then the system will create a question about the data type related to the values in the code snippet. Otherwise, a random question relating to data types will be generated.

Component Diagram



The tutor bot system will have three main sub-components, the event listener, the snippet processor, and the question handler. The event listener will handle communications with slack via the Slack API. It will be able to post messages or greetings to the tutor bot channel on slack. It will also be able to private message students for tutoring. The event listener will provide the snippet processor with code snippets sent in private messages. The question handler will provide the event listener with prompts and questions. It will also evaluate answers provided by the event listener and if needed offer feedback documentation.

Function Decomposition



The tutor bot's should be able to interact with the user by greeting in the channel or by sending private messages. System will prompt the user to enter a code snippet and receive it from the user and validate whether it is in the right format or not. If it was valid then the system will process the code snippet and should be able to create questions based on the lists of terms generated after the processing. Finally the private message will have functionality to evaluate the answers given by students and provide feedback and documentation links if needed.

Implementation

Before we start our coding, we’ve had a brief meeting with all group member to discuss more details of our software in coding specific way. The major components included communication through Slack’s API, Python for question processing(parsing) and generation, and a correctly formatted question CSV file. These key components would be translated into code using the component diagram we had previously created during the design stage. We decided that the best interface type between the Parser and Question handler was a two dimensional Python list data structure. Once the question was correctly parsed, the output would be passed to the question handler Python code which would access the CSV file to generate the content to be presented to the user. This content would then be passed back with the Slack API to the user interface. In order to manipulate the csv file easily we used Pandas which is a python library for big data manipulation and visualization. We pip-installed the library into our server by executing a command *sudo pip3 install Pandas* to be able to use the library.

Testing

We utilized Pythons unit testing suite to test our individual python files/functions. Each component of our program implemented in Python had its own test suite that accommodated for normal and abnormal outputs. After completing the unit testing for each Python component, we integrated the components that interacted with each other. Since the question handler took input from the parser, we tested that those two functions worked correctly together. Additionally, we had to make sure that the parser could take and process string input from the Slack API. We noticed that the string input from Slack did not include tab characters (‘\t’), so we altered our **/code to count the number of spaces that started a line rather than the number of tabs.

Master Test Plan

1. Introduction

    a. **Brief Description of Tutor bot system**

We are developing a system that can read code snippets that will be provided by users and give feedback, and ask questions based on the code snippets. We will integrate this system onto the slack channel we created: Rollins CMS Tutoring. The system is a web application. The bot should be able to identify if the input from the user is in the right format or not. If it was in the right format, the bot should be able to create questions based on the code snippet that users provide and display those questions to the user. In addition to that, the system should be able to validate the user’s answer and provide them some article where they can review their understanding

    b. **Objective of the test plan**

The objective of the test plan is to help us brainstorm and outline the validation of our system which will prove that our system is bug free and does what it was specified in the requirements specification document by looking at our set of requirements and the system design.
Testing will also help us facilitate quicker and smooth transition between coding activity and testing activity in our implementation phase.

c. Method of Testing

We will work our way up from the very low level of abstraction of the system. We will undergo from Unit Testing of our function in each system component to System Testing through Component Testing and Interface Testing.Since we are implementing in Python, they have a built-in unit testing library called unittest and we will use that library to perform our unit testing

2. Overall Plan

   a. Milestones of the test

We want to set the milestones to different types of the test at the different levels of abstraction of the system: unit testing, component testing, interface testing, system testing. Instead of setting milestones based on requirements and such, we can make sure that our product is reliable at each level of abstraction of architectural design and we could debug and make just-in-time fixes that would less likely cause issues later on in our development.

   b. Test Materials

      i. Test scenarios

There are two test scenarios we want to cover: the typical and atypical. In the typical scenario, the user enters their code snippet in a correct format that is with a use of triple backquote and the system will go through the entered code snippet and process them to create a set of questions and prompt the user to answer those questions. Questions will be a multiple choice format. In a typical scenario, a user enters their code snippet in an incorrect format without a triple backquote and the system will prompt the user to correctly enter their code snippet with a triple backquote.

ii. Unit Testing and its test cases.

- Event Listener will have functions to react to users messages.
  - Test case 1: the user correctly enters the input using the right format. In this test case, the system should correctly be able to store the input as string
  - Test case 2: the user incorrectly enters the input without using three backquotes. The system should display a message to prompt users to enter their input in the right format.
  - Test case 3: if the system can correctly display the questions.
- Question Handler will have a function: generate_question, to create a question and a function: generate_question_set, to create a set of question
  - Test case 1: User's input contains values such as string or numbers, check if the question handler's generate question can correctly create a question on data type
  - Test case 2: User's input contains python keywords, check if the question handler's generate question can correctly create a question on syntax
  - Test case 3: User's input contains invalid syntax, check if the question handler can correctly create a question on syntax.
  - Test case 3: User's input is empty, does not contain anything inside the triple backquote, we want to check if question handler can correctly create a generic question
  - Test case 4: check if the system can create four or seven questions

  - Test case 5: check if the set of questions generated are unique and not the same questions.
- Snippet Processor / Parser
  - Test case 1: Given a python code snippet, check if the processor process them into a two dimensional list (checking if the components returns a two dimensional list at all)
  - Test case 2: Given an empty code snippet, check if the processor create an empty two dimensional list (checking if processor returns an empty two dimensional list given an empty string)
  - Test case 3: Given a python code snippet, process and correctly create a two dimensional list
  - Test case 4: Given an empty snippet, recognize that it is empty and recognize it as wrong or no input
  - Test case 5: Given a snippet that has input parameters, make sure the parser properly separates the non-numerical or alphabetic components
  - Test 6: Test is_valid function to make sure it returns false for empty input
  - Test 7: Test is_valid function to make sure it returns false for triple quote comment
  - Test 8: Test is_valid to make sure it returns true for normal input

Interface Testing

We have several interfaces we want to test to make sure that they work as we desired and designed in the design phase.

- Interface between Event Listener and Snippet Processor

This is an interface: code snippet, that is provided by Event Listener and required by Snippet Processor to process the users' input

- Test case 1: We want to test the valid interface parameter with regular string with correct code syntax.
- Test case 2: We want to test the invalid interface parameter with empty string to see how Snippet Processor will react.
- Test case 3: We want to test the invalid interface parameter with invalid python syntax to see how Snippet Processor will react.
- Interface between Snippet Processor and Question Handler

The interface is the list of terms that is provided by the snippet processor and required by the question handler. The type of interface that snippet processors provide is procedural interface. because the Snippet processor encapsulates a set of procedures to be called by Question Handler.

- Test case 1: We want to make sure there is no interface misunderstanding, so we want to test, if the service provided by the snippet processor actually create what Question Handler expects: a two dimensional list
- Interface between Event Listener and Question Handler

There are a set of interfaces defined between these components. We have an interface: user's answer, which is provided by Event Listener and required by Question Handler and two interfaces: questions and validation of the answer of the user which are provided by Question Handler and required by Question Handler.

System Testing

We will integrate all of the system components and check two test cases

- Test case 1: Given an input that is correctly formatted by a user, check if system will process their input and generate questions, and get the user's answers to the questions and give them a feedback
- Test case 2: Given an input that is not correctly formatted by a user, the system recognizes the wrong format of the input and prompt user to re-enter their input. System will not process the wrongly formatted input

3. Procedure Control / log of the test

Dec 8th Unit testing on functions in tutorQuestion

Checked a function called check_questions which is a helper function used to identify the id of the questions already generated and stored in a list of questions. This function would help us track which questions we've created and thus avoid overlaps of questions.

tested the case where there has been no questions generated. and one question generated and more than one question generated. Test was successfully passed.

Dec 9th Unit testing on functions in tutorQuestion

Tested other functions in Tutor_Question.py check_indent function, create_iData_type question and generate_questions

I was able to successfully pass the test for check_indent function. Test indicated some None Data type error (kind of like a null pointer Error in Java) when I was working with create_iData_type because I was reassigning an updated list even though the function .append is a void function that does not return the updated list. So I was glad that I was able to point out the error I made in coding. Since this application is focused on 120 students and I've had access to their course material and by observing the material, I did not include the data types that they have not learned in the course.

Later on the same day, I created the test to test generate_questions which I checked if the function was able to create any three types of questions: iData Types, Data Types, Syntax based on the input they received. I focused on those three not including Generic because Generic questions are fun questions that we randomly ask to users. Given a two dimensional list of terms that represented a variable declaration and assignment, generate_questions successfully created an iData Types question and passed the test for the test case which involved creation of iData Type with generate_questions function. I also created two tests for Data Types and Syntax to make sure each type of questions were generated by the generate_questions function by providing them a two dimensional list of terms that were appropriate. Two tests were successfully passed.

Dec 8th Unit testing on functions in tutorParser

Tested the callSplit method. It is a helper method that splits given input by the newline character. Those lines were then passed on to the parser method and the results were stored in a 2D array and returned. I was able to test for varying inputs - an empty argument, typical input with more than one line of code, and a test to see if the method would consider parameters from the input code as a single element. Initially, the test indicated a trailing comma error which prevented it from being run. After tracing the test, a comma was found at the end of the import line and it did not belong. Getting rid of that comma allowed the tests to run.

However, the tests failed because the initial callSplit method was not written to with parameters and the tests were reliant on parameters being passed to the method. I added that in the original code and the tests passed successfully.

Dec 10th Unit testing on tutorParser

Tested the is_valid method. The method should verify that the user input is in the correct format i.e. no empty input or if the code began with triple back quotes, which represent code blocks in Slack. I tested for empty input, input code with triple back quotes, and normal input. All tests ran and passed successfully.

Tested the par method. This method takes a line from the callSplit method and parses them by separating special characters, variable and method names, keywords (like def, print, etc.), numbers, and adds them to a single array and returns it back to the callSplit method. The method also recognizes comments "#" and does not include them in the array. The tests performed on this method include a test with empty input, which should return an empty array, test with a typical code block, which should return a 2D array of the code block, and a test to filter out comments, which return a 2D array that does not include the comment line. All tests ran and passed successfully.

Deployment

Our project was hosted on a DigitalOcean droplet server. It was running Ubuntu 18.04 with 1GB of memory and 25GB of disk space. Our small Python3 scripts would not over-tax the system – the main function it provided was availability. The cloud server was free through a GitHub student resources collection. We created user accounts with sudo rights for the group and for Dr. Elva to access the server. We configured the firewall for our scripts to be able to communicate through the Slack API. Our code was managed and delivered to the server via GitHub. Private Slack authentication information was omitted from the GitHub repo for security reasons. After the repo was cloned or updated the scripts and other relevant files would be moved to their locations on the server and the Slack authentication information would be added back into the files. The tutor system was launched through a bash script that opened the sever firewall and launched the script that listened for Slack events.

Division of Labor

As we went on to the implementation phase, we divided our labor by components. Processor/Parser was developed by Jenny and Deandra, Question Handler was developed by Hiro and Event Listener and server setup were developed and prepared by Noah.

Lessons Learned

This project showed us the importance of testing, clear communication in the form of documentation, and appropriate implementation time. As we have discussed in class, testing is a critical part of the SDLC that occurs throughout multiple phases. Specifically, testing during implementation is key in making timely fixes and aiming for exhaustive testing. Because we had a shortened implementation time, we did not have the time we would have liked to ensure that our test cases were complete. For instance we were not able to complete the interface testing and system testing after the completion of Unit testing because we did not have enough time to explore tools we could use to test them.

In relation to communication, our project required us to work with multiple platforms (Python IDE and Slack API) which necessitated that we have clear communication and documentation for how these technologies worked and interacted with each other. These interactions included the passing of parameters between functions/platforms, reading from a CSV file, and taking user input. Having clear documentation of the expected input/output of each function allowed different programmers to code interrelated functions without ambiguity. Clear documentation also enabled us to easily translate our diagrams (component and use case) and functional requirements into code. As we learned about in class, the outputs of the previous phases of the SDLC are meant to facilitate future phases, and we benefitted from that by creating and using our documentation during implementation.

The final major takeaway was the importance of allowing enough time to have a thorough implementation process. As previously discussed, we would have liked to have been able to have more time to conduct more testing during implementation. More thorough testing resolves issues in real-time before they can become more costly further in the SDLC.

Additionally, if we had more time for implementation we would have more easily been able to execute all of the intended features of our software. We also may have been able to create a larger set of questions to diversify the potential content for students.

Overall, this project embodied many of the key concepts we had discussed in class. Having the opportunity to apply these concepts to a project reinforced the concepts of testing, communication, documentation, and time constraints. We also experienced the challenges of system heterogeneity, shortened development time, and integrating components from different programmers. Because of the content we had covered in class, we were aware that these are common difficulties during software engineering. This allowed us to be conscious of these possible constraints and attempt to work around them.

Source Code

tutorBot.py

```
#!/usr/bin/python3



##############################################################################################################################

#  NAME:  Jenny Goldsher, Noah Harvey, Deandra Martin, Hiroki Sato

#  DATE:  05112020

#  IDEA:  this script will listen for a message and respond

##############################################################################################################################



####  IMPORTS  ##############################################################################################################



from random import random

import slack

from flask import Flask

from slackeventsapi import SlackEventAdapter

from tutorParser import callSplit

from tutorQuestions import generate_questions



####  GLOBALS  ##############################################################################################################



token = ""

secret = ""

#responses = ["Hey!", "Hi!", "What's up?", "Does this syntax make me look fat?", "That's Tudor to you!", "Genaric response"]

conversations = {}



####  FUNCTIONS  ############################################################################################################



def questionUnpacker(q):

    qText = q["Question"]

    if(str(q['A']) != "nan"):

        qText += ("\nA. " + str(q['A']))

    if(str(q['B']) != "nan"):

        qText += ("\nB. " + str(q['B']))

    if(str(q['C']) != "nan"):
        qText += ("\nC. " + str(q['C']))
    if(str(q['D']) != "nan"):
        qText += ("\nD. " + str(q['D']))
    return(qText)


####  MAIN  ##################################################################################################################


server = Flask(__name__)
slackEvent = SlackEventAdapter(secret,"/slack/events",server)   #  authenticates provides url for events
bot = slack.WebClient(token=token)
botID = bot.api_call("auth.test")["user_id"]                    #  save bot id so the bot knows if a message is its own


@slackEvent.on("message")                                       #  message.channels event ("event"{"type":"message"})
def message(content):
    event = content.get("event", {})
    channel = event.get("channel")
    user = event.get("user")
    text = event.get("text")

    print(content,event,channel,user,text,sep='\n')                                                   ##  DEBUGGING  ##


    if(user != botID):
        if(user not in conversations):                          #  first message/welcome
            welcome = "Welcome to the tutor bot - message code blocks to generate questions"
            bot.chat_postMessage(channel=channel,text=welcome)
            conversations[user] = [False,[],[text,welcome]]
        elif("```" in text):                                    #  code block in message
            textSplit = text.split("```")
            if(len(textSplit) == 3):                            #  single code block
                codeLines = callSplit(textSplit[1])             #  parse code bloack with tutorParser
                #print(codeLines)                                                                      ##  DEBUGGING  ##
                conversations[user][1] = generate_questions(codeLines)          #  geneate and save list of questions
                if(len(conversations[user][1]) > 0):            #  tutorQuestions was able to generate questions
                    question = conversations[user][1].pop(0)    #  if there are questions pop the first one into question
                    conversations[user][0] = question["Answer"] #  set the answer in user conversation value
                    questionText = questionUnpacker(question)   #  unpack question text into string
                    print('',conversations[user][0],question,'',sep='\n')                            ##  DEBUGGING  ##
                else:                                           #  tutorQuestions failed to generate questions
                    conversations[user][0] = 'A'                                #  for debugging (really want ans letter)
                    questionText = "test question (answer A) A. B. C. D."       #  for debugging (really want question text)
                conversations[user][2].append(text)
```

```
        conversations[user][2].append(questionText)
        bot.chat_postMessage(channel=channel,text=questionText)
    else:                                               #  unexpected number of code blocks
        textError = "I'm sorry, I couldn't find your code"
        conversations[user][2].append(text)
        conversations[user][2].append(textError)
        conversations[user][0] = False
        bot.chat_postMessage(channel=channel,text=textError)
elif(conversations[user][0]):                           #  value (or, not False) in index 0 (answer)
    if(conversations[user][0] in text.upper()):         #  answer letter in message text
        correct = "That's correct!"
        conversations[user][2].append(text)
        conversations[user][2].append(correct)
        bot.chat_postMessage(channel=channel,text=correct)
        if(len(conversations[user][1]) > 0):            #  if there are still questions to ask
            question = conversations[user][1].pop(0)
            conversations[user][0] = question["Answer"] #  set the answer in user conversation value
            questionText = questionUnpacker(question)   #  unpack question text into string
            print('',conversations[user][0],question,'',sep='\n')                                     ##  DEBUGGING  ##
            conversations[user][2].append(text)
            conversations[user][2].append(questionText)
            bot.chat_postMessage(channel=channel,text=questionText)
        else:                                           #  no questions left to ask
            prompt = "Post a code block to generate questions"
            conversations[user][2].append(text)
            conversations[user][2].append(prompt)
            conversations[user][0] = False
            bot.chat_postMessage(channel=channel,text=prompt)
    else:
        incorrect = "I'm sorry, I don't think that's right."
        conversations[user][2].append(text)
        conversations[user][2].append(incorrect)
        bot.chat_postMessage(channel=channel,text=incorrect)
        if(len(conversations[user][1]) > 0):            #  if there are still questions to ask
            question = conversations[user][1].pop(0)
            conversations[user][0] = question["Answer"] #  set the answer in user conversation value
            questionText = questionUnpacker(question)   #  unpack question text into string
            print('',conversations[user][0],question,'',sep='\n')                                     ##  DEBUGGING  ##
            conversations[user][2].append(text)
            conversations[user][2].append(questionText)
            bot.chat_postMessage(channel=channel,text=questionText)
        else:                                           #  no questions left to ask
            prompt = "Post a code block to generate questions"
            conversations[user][2].append(text)
            conversations[user][2].append(prompt)
```

```
                conversations[user][0] = False

                bot.chat_postMessage(channel=channel,text=prompt)

        else:                                                 #  no code block in message, or question posed - prompt

            prompt = "Post a code block to generate questions"

            conversations[user][2].append(text)

            conversations[user][2].append(prompt)

            conversations[user][0] = False

            bot.chat_postMessage(channel=channel,text=prompt)



        #bot.chat_postMessage(channel=channel,text=(responses[int(random()*len(responses))]),)




# @slackEvent.on("app_mention")

# def mention(content):

#     event = content.get("event", {})

#     print(content,event,sep='\n')



#     channel = event.get("channel")

#     user = event.get("user")



#     bot.chat_postEphemeral(channel=channel,text="this is a response to a message",user=user,blocks=block)




# @slackEvent.on("message.app_home")

# def privateMessage(content):

#     event = content.get("event", {})

#     print(content,event,sep='\n')



#     channel = event.get("channel")

#     user = event.get("user")



#     bot.chat_postEphemeral(channel=channel,text="this is a response to a message",user=user,blocks=block)



if(__name__=="__main__"): server.run()                        #  might need to change debug to False
```

tutorQuestions.py

```
#!/usr/bin/python3


################################################################################################################################
#  NAME:  Jenny Goldsher, Noah Harvey, Deandra Martin, Hiroki Sato
#  DATE:  09112020
#  IDEA:  opens questions file and generates questions
#       There are certain types of questions that must be created depending on the syntax error that they make or
#       what they implement within the codeblock. For example, syntax question about the indentation is always created when user
#       used wrong indentation.


################################################################################################################################


####  IMPORTS  #################################################################################################################
import pandas as pd
from random import randint, choice
from ast import literal_eval


####  GLOBALS  #################################################################################################################
question_df = pd.read_csv('Questions_11_27.csv')
existing_q_id = []
syntax_list = ['def','if','else','==','+','-','*','/',"'",'"']
data_types = ['int','str','float','bool','list']
                                                                                                   # we are using a python libary called
Pandas to manipulate the
                                                                                                   # csv file easily by creating a pandas
dataframe.
                                                                                                   # .set_index would make one of the column
in the csv file
                                                                                                   # 'Question Type' as index and we can
extract the matching
                                                                                                   # question according to the input we'll
receive




####  FUNCTIONS  ###############################################################################################################
################################################################################################################################
#### check_question ############################################################################################################
```

```
# Input: list of questions
# Output list that contains question id
# Objective: iterate through the list of questions created and record the existing_question_id so that we can keep track of what questions
can be created and
# what questions should not be created



def check_question(questions):

    existing_q_id = list()

    if len(questions) == 0:                                                         # if the questions list is empty, return a
list with 0
        return [0]
    else:                                                                           # if the questions list is not empty,

        for q in questions:                                                         # iterate throught the list of questions
            existing_q_id.append(q['id'])

    return existing_q_id




##########################################################################################################################
#### check_indent function ###############################################################################################



# Input: two dimensional list
# Output: boolean
# Objective: iterate through the two dimensional list and make sure there is no error with indentation
# Returning True indicate that there is no indentation issue, and False represents an issue with spaces



def check_indent(lines):

    # local variable flag, if anything weird didn't happen then this function will return True
    flag = True

    for line in range(len(lines)):                                                  # iterating through the 2-d list

        cur = lines[line]
        if (cur[0] % 4 != 0):                                                                  # if the number of space is not multiple
of 4


            flag = False                                                                       # this is the base case that can be
applied to either a single
            break                                                                              # line of code or multiple lines of code.


        if len(lines) > 1:                                                                     # if the code was not a single line.

            # get the previous lines
            prev = lines[line - 1]

            if ((prev[len(prev)-1] == ':') and (cur[0] != (prev[0] + 4))):                     # the previous line ends with : but the
number of space is not
                                                                                               # incremented by 4
                flag = False
                break


    return flag


################################################################################################################################
#### create_iData_type function ################################################################################################


# Input: list which is the entire row in the two-d list and an int which is an index of '='
# Output: dictionary which is a question, and choices and its answer and link to the feedback
# Objective: create a iData type question


def create_iData_type(line, index):

    # get the iData_type question as dictionary
    question = question_df.set_index('Question Type').loc['iData Types'].to_dict()
    question_df.reset_index()
    print(question)                                                                            # debugging
    choices = data_types
    answer = question['Answer']
    print(choices)                                                                             # debugging
    # list to keep track of data type that was chosen as the
    chosen = []
```

```
# getting the variable name and getting the data_type of variable.
variable_name = line[index - 1]
variable_type = type(literal_eval(line[index + 1]))


# print(variable_name, variable_type)                                                   # debugging

if variable_type == str:


    # modifying the question choice to avoid overlap
    answer = 'str'



if variable_type == int:
    answer = 'int'




if variable_type == float:
    answer = 'float'

if variable_type == bool:
    answer = 'boolean'



if variable_type == list:
    answer = 'list'

print("Answer", answer)

# using ascii value, selecting the key to put the answer and place the answer.
answer_key = chr(65 + randint(0,3))
chosen.append(answer)
question[answer_key] = answer
question['Answer'] = answer_key
print("Printing chosen", chosen)

# selecting the rest of the answers.
for i in range(3):


    key = chr(65 + randint(0,3))
```

```
        while key == answer_key:
            key = chr(65 + randint(0,3))


        dummy = choice(choices)

        while dummy in chosen or dummy == answer:
            dummy = choice(choices)

        chosen.append(dummy)
        question[key] = dummy


    # modify the question sentence
    question['Question'] = question['Question'].replace("(variable name)",variable_name)

    return question



################################################################################################################################


"""This function is going to create a set of questions  """


# Input: two dimensional list called term_lists
#        the first index, the row represents the line, second index columns are the terms on that line.
# Output: two dimensiocal list which contains a list of questions and answer keys.


def generate_questions(term_lists):

    questions = list()
    iData_type = False
    Generic = False

    # have a for loop to create randomly selected 7 questions and append to the questions list
    # first we could check for line indentation which is the very important and simple syntax question we can ask
    indentation = check_indent(term_lists)

    # if we see the flag being False, append the syntax question about the indentation/spaces
    if (indentation == False):
        q = question_df.set_index('Question Type').loc['Syntax'].iloc[4].to_dict()
        questions.append(q)                                                                           # set_index is going to use certain column
as index of the
                                                                                                      # pandas dataframe which is id in this
case, and we know the
                                                                                                      # id for the indentation syntax so we get
the question by
                                                                                                      # loc[] operator and preserve the entire
row as a list.
    # going through the two dimensional list again
    # we will visit each line to create data type question, or syntax question, or generic question



    for line in range(len(term_lists)):
        # question = generate_question(questions, tokens)
        # we first want to make sure what questions are already in the list of questions
        existing_q_id = check_question(questions)
        # get the current line of code
        current = term_lists[line]
        print('Current line:', current)
        # we will have an internal for loop to go through each terms except the first elements which indicates the number of spaces.

        for term in range(1,len(current)):

            t = current[term]

            if (t == '=' and iData_type == False):
                # create iData_type_qestion
                question = create_iData_type(current, term)
                iData_type = True
                questions.append(question)
                continue

            if (t == '=' and iData_type == True):
                # print('Creating data type question')
                # create a data type question.

                dtq = question_df.set_index('Question Type').loc['Data Types']
                question_df.reset_index()
                question = dtq.iloc[randint(0, len(dtq) - 1)]



                while question['id'] in existing_q_id:
                    question = dtq.iloc[randint(0, len(dtq) - 1)]

                    q = question.to_dict()
                q = question.to_dict()
                questions.append(question)
                continue

            if (t in syntax_list):

                # create syntax question

                stq = question_df.set_index('Question Type').loc['Syntax']
                question_df.reset_index()
                question = stq.iloc[randint(0, len(stq) - 1)]

                while question['id'] in existing_q_id:
                    question = stq.iloc[randint(0, len(stq) - 1)]


                q = question.to_dict()
                questions.append(question)
                continue




        # check if we reached the number of questions we wanted.

        if len(questions) == 4:
            break

    # after going through everything and you don't have enough question, add one generic question.
    if len(questions) < 4:

        gnq = question_df.set_index('Question Type').loc['Generic']
        question_df.reset_index()
        question = gnq.iloc[randint(0, len(gnq) - 1)].to_dict()
        questions.append(question)

    return questions




####  MAIN  #####################################################################################################################
```

```
def main():
    print(questions)



if(__name__=="__main__"): main()
```

tutorParser.py

```
#!/usr/bin/python3



##################################################################################################################################
#  NAME:  Jenny Goldsher, Noah Harvey, Deandra Martin, Hiroki Sato
#  DATE:  09112020
#  IDEA:  holds parsing funcion
##################################################################################################################################




####  FUNCTIONS  ################################################################################################################


def is_valid(inp):                                          # this function takes in an input which is the string that bot receives from user
and                                                                # validates if they used a code block or not
    valid = True

    if len(inp) == 0:
        return False

    elif inp[0:3] != "```":
        return False

    return valid                                            # Output: Boolean value, True if they used the codeblock which is the correct
format. False                                                      # if they did not use the correct Format.
                                                            # By the string slicing and checking the first three character being the triple
backquote or not will determine the format which the user used.




def par(inp):                                               # Function takes a line to parse and return as an array
    str1 = inp.split()                                      # Split the line by spaces
    length = len(str1)
```

```
if length == 0:                                         # Returns an empty list if input is empty
    return

arr2 = []
count1 = 0
digit = 0
for element in str1:
    if element.isalpha() or len(element) <= 1:
        arr2.append(element)
    elif len(element) > 1:

        count1 = 0
        quote = 0
        digit = 0
        has_fp = 0

        for x in range(len(element)):
            has_underscore = 0
            has_number = 0

            if element[x] == '_':
                has_underscore +=1
            elif element[x] == '"' or element[x] == "'":
                quote +=1
            elif element[x].isdigit():
                has_number += 1
                digit += 1
            elif element[x] == '.' and digit != 0:
                has_fp += 1
            elif element[x].isalpha():
                count1 +=1
        if has_number + has_underscore + count1== len(element):
            arr2.append(element)
        elif quote + count1 == len(element):
            arr2.append(element)
        elif has_fp + digit == len(element):
            arr2.append(element)
        else:
            store = 0
            for f in range(len(element)):
                store+=1
                if element[f].isalpha() == False and element[f].isdigit() == False and element[f] != '_':
                    store = f
                    break
            if store != 0:
                    word = ""
                    for v in range(0,store):
                        word = word + element[v]
                    arr2.append(word)
                    for a in range(store,len(element)):
                        arr2.append(element[a])



    if str1[0] == "#":                                      # Disallow comments being parsed by returning
        return

    count = 0                                               # Count all of the spaces and then subtract non starter
    for i in range(len(inp)):                               # spaces to look at line indentation
        if inp[i] == " ":
            count+=1
    count = count - len(str1)+ 1

    li = [count]                                            # Make another array that has the count of spaces
    for j in arr2:                                          # then add the parsed line
        li.append(j)
    return li



def callSplit(txt):
    arr = []                                                # Array to store each line array
    lines = txt.split("\n")                            # Make an array where each value is one line

    for i in lines:                                         # Pass each line to the parser, which will return its value
        if par(i) != None:                                  # Populate 2D array with lines without including 'None' values
            arr.append(par(i))

    #print(arr)
    return arr



####  MAIN  ####################################################################################################################


def main():
    #inputtxt = "def function():\n     x = 'hello'"              # TEST
    #inputtxt = "''' testing comments\nx = 15"              # TEST
    inputtxt = "def func_one():\n    x = 15\n     y = 12 + x\n  for i in range(10):"                                       # TEST
    #inputtxt = "x = 'hello'"

    # inputtxt = (input())

    print(callSplit(inputtxt))



if(__name__=="__main__"): main()
```