**PROJECT REPORT**

**ON**

**CHATBOT USING PYTHON**

# Submitted in partial fulfillment of requirements to

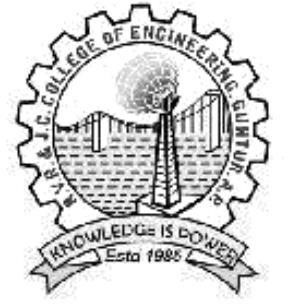
IT 353– MINI PROJECT

BY

P.SUJITHA(Y22IT092)

M.AASWITHA(Y22IT082)

P.NITHISHA(Y22IT090)



**OCTOBER 2024**

**R.V.R & J.C.COLLEGE OF ENGINEERING(AUTONOMOUS)**

**(NAAC A+ GRADE) (Approved by A.I.C.T.E) (Affiliated to Acharya Nagarjuna University)**

**Chandramoulipuram : : Chowdavaram GUNTUR – 522 019**

# R.V.R & J.C.COLLEGE OF ENGINEERING

**DEPARTMENT OF INFORMATION TECHNOLOGY**

## BONAFIDE CERTIFICATE

This is to certify that this project work titled CHATBOT is the bonafide work of (P.sujitha(y22it092) , m.aaswitha(y22it082) , p.nithisha(y22it090)) who have carried out the work under my supervision, and submitted in partial fulfillment of the requirements to **IT-353, MINI PROJECT** during the year 2024-2025.

**Dr.A.Yaswanth Kumar Dr. A.Srikrishna**

Lecturer Incharge Prof.&HOD, Dept. of IT

## ACKNOWLEDGEMENTS

The successful completion of any task would be incomplete without a proper suggestions, guidance and environment. Combination of these three factors acts like backbone to our Project

**“CHATBOT USING PYTHON”.**

We are very much thankful to **Dr.Kolla.Srinivas**, Principal of R.V.R. & J.C. College of Engineering, Guntur for having allowed delivering this mini project.

We express our sincere thanks to **Dr.A.Srikrishna**, Professor, Head of the Department of Information Technology for her encouragement and support to carry out this mini project successfully.

We are very glad to express our special thanks to **Dr.A.Yaswanth kumar**, Assistant Professor in Department of Information Technology for timely, guidance and providing us with most essential materials required for the completion of this report.

Finally we express our sincere thanks to all the **Teaching** and **Non-Teaching staff**

of **IT Department** who have contributed for the successful completion of this report.

P.SUJITHA(Y22IT092)

M.AASWITHA(Y22IT082)

P.NITHISHA(Y22IT090)

### CONTENTS

**Chapter No. & Name Page No.**

1. Problem statement. 1
2. SRS Documentation - Requirements elicitation. 2
3. System Requirements Specification 3
4. Requirements modeling 4
5. Identification of Actors, Use cases. 5
6. Construction of Use case diagram and flow of events 10
7. Building a Business Process model using UML activity diagram 13
8. Construction of Sequence diagram 20
9. Construction of Collaboration diagrams 24
10. Construction of UML Class diagram 26
11. Analyzing the object behavior by constructing UML State Chart diagram. 31
12. Construction of implementation diagrams. 34
13. Sample application code and Output Screeshots 36
14. Testing 43
15. Conclusion 44

Reference

**1.PROBLEM STATEMENT**

#### CHATBOT

In our project we explore how a chatbot can give information to students about school-related information. In the first iteration of the project we created a chatbot for giving students information about where to get coffee etc. at IFI. One of our hypothesis was that information given by chatbots would be useful for new students at IFI, giving them information about things that we consider to be important when you’re a first year students. In the second iteration we wanted to explore the use of chatbots through theory and used this in combination with testing to learn more about how a chatbot for this context should be. In the final iteration, iteration three, we improved and changed the chatbot based on the results from the last iteration and made a plan for evaluate the chatbot. The plan was then executed with five participants. In our conclusion we discuss the results from the evaluation in the light of our research question.

#### 2.SRS DOCUMENTATION - REQUIREMENTS ELICITATION

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Details** | **Functionalities** | **Priorities** |
| R1 | Chatbot remains name and created year | Functional Data | Must have |
| R2 | Chatbot prompts the user to specify the name | Functional Data | Must have |
| R3 | Chatbot guess the age of user | Functional Data | Must have |
| R4 | Chatbot counts the number | Functional Data | Must have |
| R5 | Chatbot conducts tests | Functional Data | Must have |

**1. SYSTEM REQUIREMENT SPECIFICATION**

**Software Requirements:**

* Operating System : windows 11

* Coding language : PYTHON

**Hardware Requirements**:

* Personal computer with keyboard and mouse maintained with uninterrupted power supply.
* Processor : Intel® core™ i5

* Installed Memory (RAM) : 1.00 GB

* Hard Disc : 40 GB

#### 3.REQUIREMENTS MODELING

The most important factor for the success of an IS project is whether the software product satisfies its users' requirements. Models constructed from an analysis perspective focuses on determining these requirements. This means Requirement Model includes gathering and documenting facts and requests.

The use case model gives a perspective on many user requirements and models them in terms of what the software system can do for the user. Before the design of software which satisfies user requirements, we must analyze both the logical structure of the problem situation, and also the ways that its logical elements interact with each other. We must also need to verify the way in which different, possibly conflicting, requirements affect each other. Then we must communicate this under standing clearly and unambiguously to those who will design and build the software.

Use-case diagrams graphically represents system behavior (use cases). These diagrams present a high level view of how the system is used as viewed from an outsider’s (actor’s) perspective. A use-case diagram may contain all or some of the use cases of a system.

A use-case diagram can contain:

* ·actors ("things" outside the system)
* ·use cases (system boundaries identifying what the system should do)
* interactions or relationships between actors and use cases in the system including the associations, dependencies, and generalizations.

Use-case diagrams can be used during analysis to capture the system

requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implement

#### 4.IDENTIFICATION OF ACTORS and USECASES

**Identification of ACTORS :**

Actors represent system users. They are NOT part of the system .They represent anyone or anything that interacts with the system.

An actor is someone or something that:

* Interacts with or uses the system
* Provides input to and receives information from the system
* Is external to the system and has no control over the use cases

Actors are discovered by examining:

* Who directly uses the system
* Who is responsible for maintaining the system
* External hardware used by the system
* Other systems that need to interact with the system

The needs of the actor are used to develop use cases. This insures that the system will be what the user expected.

**Graphical depiction:**

An actor is a stereotype of a class and is depicted as a “stickman” on a use-case diagram. For example,

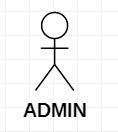


Actors identified in the information system are:

1. Admin
2. User

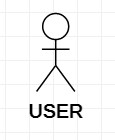
* 1. Admin:
     + Admin only can do chat with chatbot
     + Admin Can’t update information
     + Admin Can’t view information
     + Admin Can’t delete information
     + Admin Can’t add information

and he has to logout the account after the desired actions complete.



* 1. User:
     + User can do chat with chatbot
     + User can update information
     + User can view information
     + User can delete information
     + User can add information

and he has to logout the account after the desired actions complete.



#### Identification of Use-Cases Or Sub Use-Cases

Use case can be described as a specific way of using the system from a user’s perspective. A more detailed description might characterize a use case as:

* A pattern of behavior the system exhibits
* A sequence of related transactions performed by an actor and the system The UML notation for use case is:

Login

**Purpose of usecases:**

* Well structured use cases denote essential system or subsystem behaviours only, and are neither overly general nor too specific.
* A use case represents a functional requirement of the system as a whole
* Use cases represent an external view of the system
* A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system with the system itself.

#### Use-cases identified for Chatbot

##### 1 .Use-case name: CHAT

This is a use case which is used by actor to chat with the bot

##### Chat

2. **Use-case name :** VIEW INFORMATION

System allows admin to view information

View information

1. **Use-case name:** UPDATE INFORMATION

This use case allows admin to update information like updating responses,code,user interactions, etc.,

Update information

1. **Use-case name:**ADD INFORMATION

This use case allows the admin to add infromation

Add infromation

5 .**Use-case name:** DELETE INFORMATION

This use case allows admin can delete the information when it is doesn’t want

Delete information

#### Identification of RELATIONS

**Association Relationship:**

An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. If two objects are usually considered independently, the relationship is an association.

Login

**Dependency**

**Relationship:**



A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning.

We can provide here 1. Include relationship:

It is a stereotyped relationship that connects a base use case to an inclusion use case .An include relationship specifies how the behavior in the inclusion use case is used by the base use case.

Base use-case

<<

include

>>

Inclusion

use

-

case

2. Extend relationship:

It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case.

<<extend>> is used when you wish to show that a use case provides additional functionality that may be required in another use case.

<<extend>>

Print campaign summary check campaign budge **Identification of RELATIONS**

**Association Relationship:**

An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. If two objects are usually considered independently, the relationship is an association.

Login

**Dependency**

**Relationship:**



A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning.

We can provide here 3. Include relationship:

It is a stereotyped relationship that connects a base use case to an inclusion use case .An include relationship specifies how the behavior in the inclusion use case is used by the base use case.

Base use-case

<<

include

>>

Inclusion

use

-

case

4. Extend relationship:

It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case.

<<extend>> is used when you wish to show that a use case provides additional functionality that may be required in another use case.

<<extend>>

Print campaign summary check campaign budg

**5.CONSTRUCTION OF USE CASE DIAGRAM AND FLOW OF EVENTS.**

Use-case diagrams graphically represent system behavior. These diagrams present a high level view of how the system is used as viewed from an outsider’s perspective.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use usecase diagrams to specify the behavior of the system as implemented.

**USE CASE DIAGRAM FOR FRAMS:**



#### FLOW OF EVENTS

A flow of events is a sequence of transactions performed by the system. They typically contain very detailed information .Flow of events document is typically created in the elaboration phase.

Each use case is documented with flow of events

* A description of events needed to accomplish required behaviour
* Written in terms of what the system should do, NOT how it should do it
* Written in the domain language , not in terms of the implementation

A flow of events should include

* When and how the use case starts and ends
* What interaction the use case has with the actors
* What data is needed by the use case
* The description of any alternate or exceptional flows

The flow of events for a use case is contained in a document called the use case specification. Each project should use a standard template for the creation of the use case specification. Includes the following

1. Use case name – Brief Description
2. Flow of events –
   1. Basic flow
   2. Alternate flow
   3. Special requirements
   4. Pre conditions
   5. Post conditions
   6. Extension points

#### FLOW OF EVENTS

**1. Greeting:** The script starts by greeting the user, introducing itself with a provided bot name and birth year.

**2.User Name Reminder:** The script asks the user to remind it of their name and then acknowledges the user's name.

**3.Guessing Age:** The script attempts to guess the user's age by asking for remainders when dividing the age by 3, 5, and 7. It then calculates and prints the guessed age.

**4.Counting:**The script requests the user to input a number and then demonstrates its ability to count from 0 to the specified number.

**5.Programming Knowledge Quiz:**

* The script presents a series of multiple-choice questions related to programming knowledge. The user is prompted to choose an option, and the script provides feedback on the correctness of the answer.
* The quiz consists of questions such as the purpose of methods, the establishment year of a college, the founder of programming languages, the purpose of variables, and the definition of a program.

**6.Ending:** After completing the quiz, the script congratulates the user and wishes them a nice day.

**7.User Interaction:**Throughout the interaction, the script takes user input using the input() function, and it validates the user's responses against predefined correct answers.

**8.Modular Design**:The script is modularized with functions for specific tasks, such as greeting, age guessing, counting, and quizzes. This makes the code more organized and easier to maintain.

**10.End of Script**:The script ends by waiting for user input, allowing the user to take their time before closing the program.

**6.BUILDING A BUSINESS PROCESS MODEL USING UML ACTIVITY**

**DIAGRAM**

An Activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. The purpose of Activity diagram is to provide a view of flows and what is going on inside a use case or among several classes. You can also use activity diagrams to model code-specific information such as a class operation.

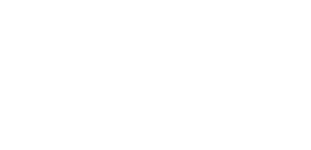
Activity diagrams are very similar to a flowchart because you can model a workflow from activity to activity. An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.

* Activity Diagrams also may be created at this stage in the life cycle. These diagrams represent the dynamics of the system. They are flow charts that are used to show the workflow of a system; that is, they show the flow of control from activity to activity in the system, what activities can be done in parallel, and any alternate paths through the flow.
* At this point in the life cycle, activity diagrams may be created to represent the flow across use cases or they may be created to represent the flow within a particular use case.
* Later in the life cycle, activity diagrams may be created to show the workflow for an operation.

The following tools are used on the activity diagram toolbox to model activity diagrams:

**Activities:**

An activity represents the performance of some behavior in the workflow.



NewActivity

**Transitions:**

Transitions are used to show the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity.



**Decision Points:**

When modeling the workflow of a system it is often necessary to show where the flow of control branches based on a decision point. The transitions from a decision point contain a guard condition, which is used to determine which path from the decision point is taken. Decisions along with their guard conditions allow you to show alternate paths through a work flow.

Decision point

**Start state:**

A start state explicitly shows the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram.



**End state:**

An End state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.

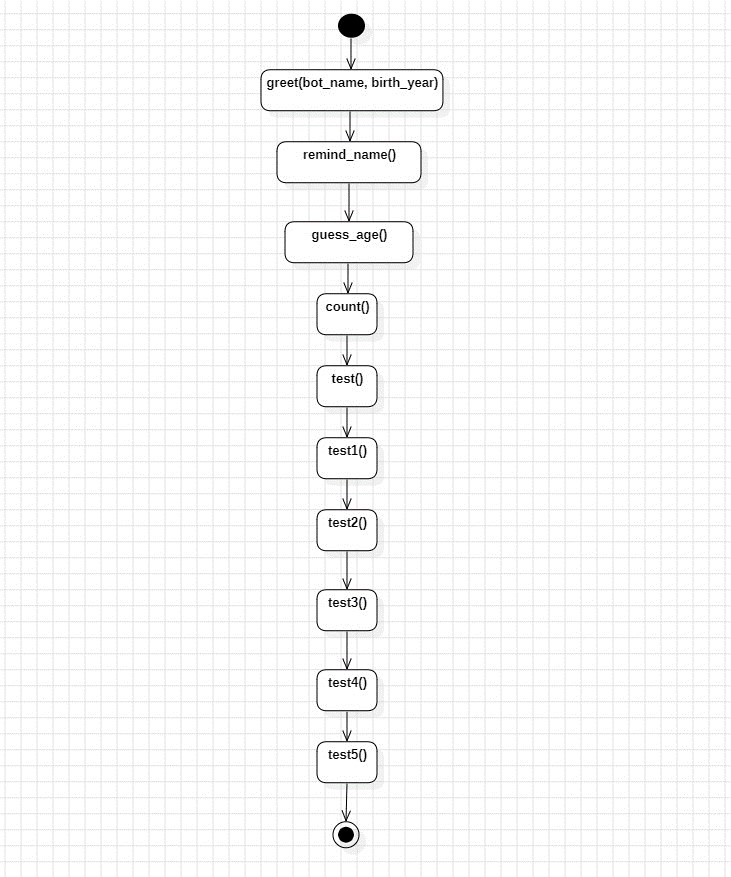


End state

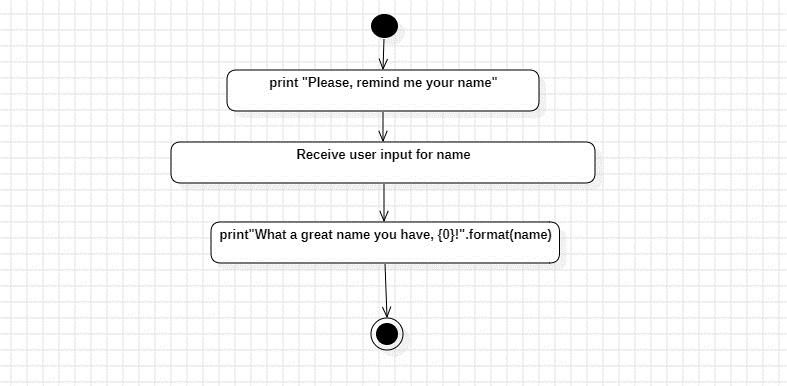
**Swim Lanes:**

Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organization is responsible for the activities contained in the swim lane.

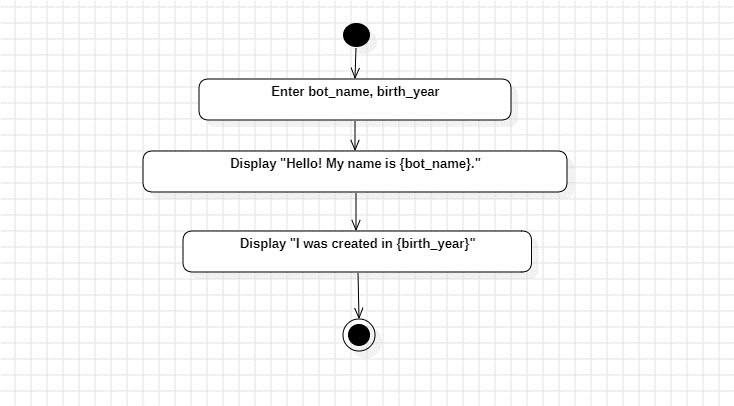
 Horizontal synchronization  Vertical synchronization.



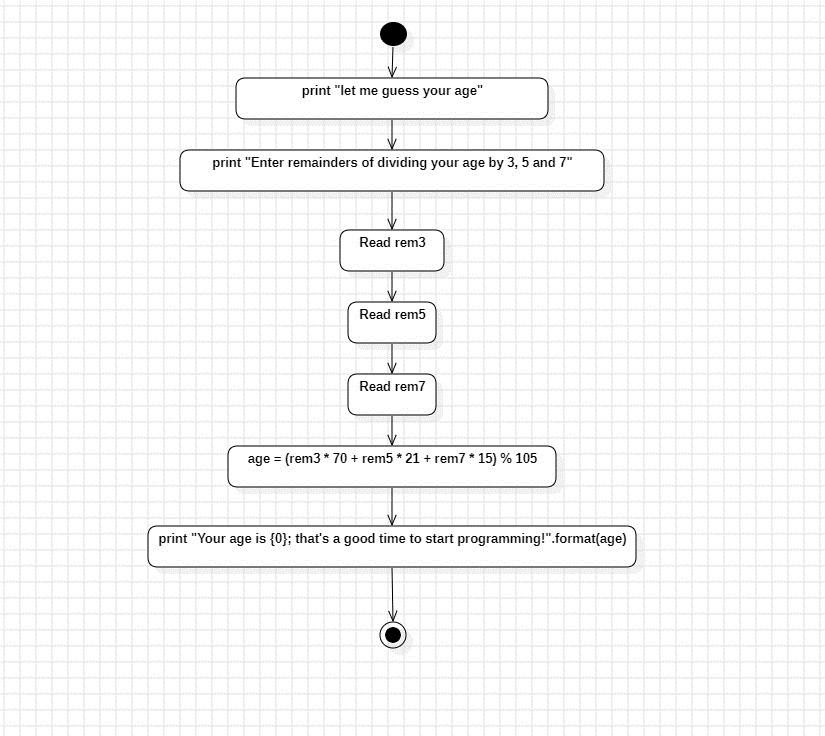
**Activity diagram for User Name Reminder**



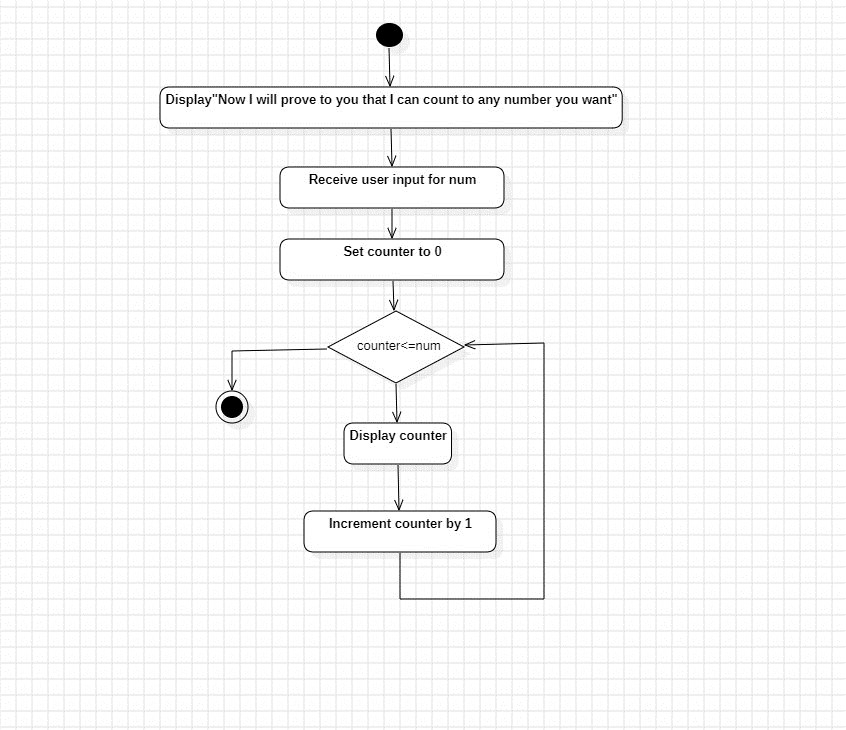
**Activity diagram for**



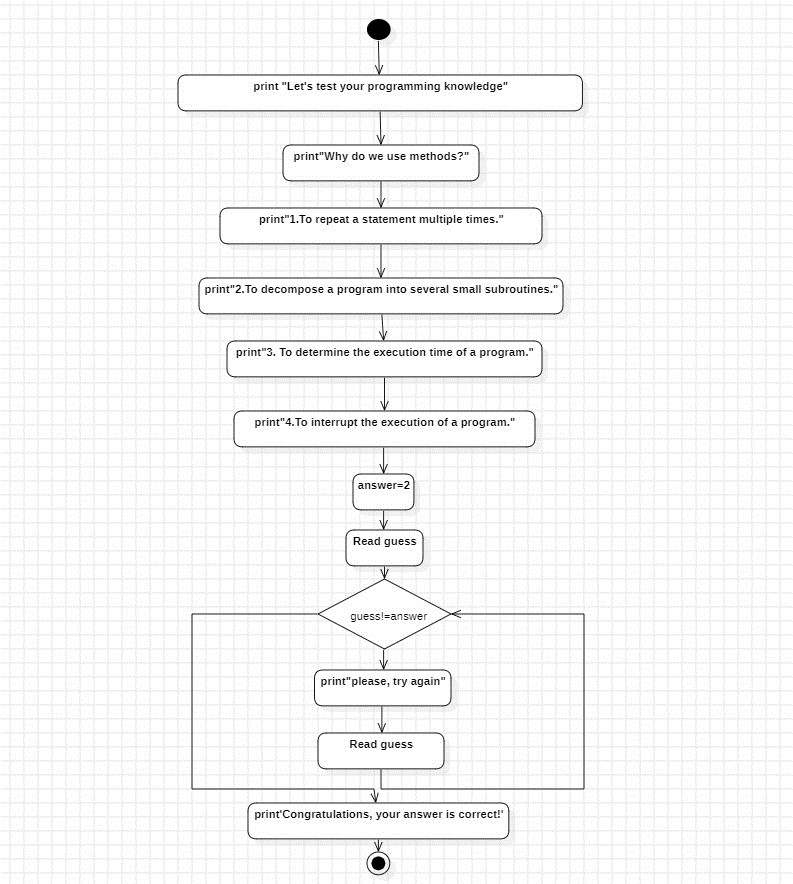
**Activity diagram for Guessing age**



**Activity diagram for counting**



**Activity diagram for Testing**



**7.CONSTRUCTION OF SEQUENCE DIAGRAMS.**

A sequence diagram is a graphical view of a scenario that shows object interaction in a time based sequence--what happens first, what happens next…

Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A sequence diagram has two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects. The vertical line is called the object’s lifeline. The lifeline represents the object’s existence during the interaction. Steps:

1. An object is shown as a box at the top of a dashed vertical line. Object names can be specific (e.g., Algebra 101, Section 1) or they can be general (e.g., a course offering). Often, an anonymous object (class name may be used to represent any object in the class.)
2. Each message is represented by an Arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Each message is labeled with the message name.

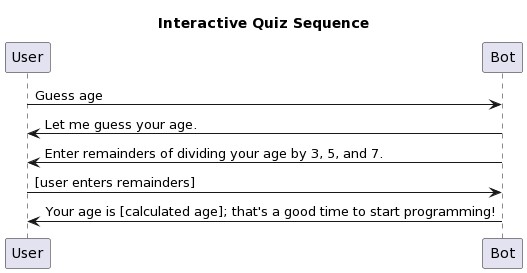
There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

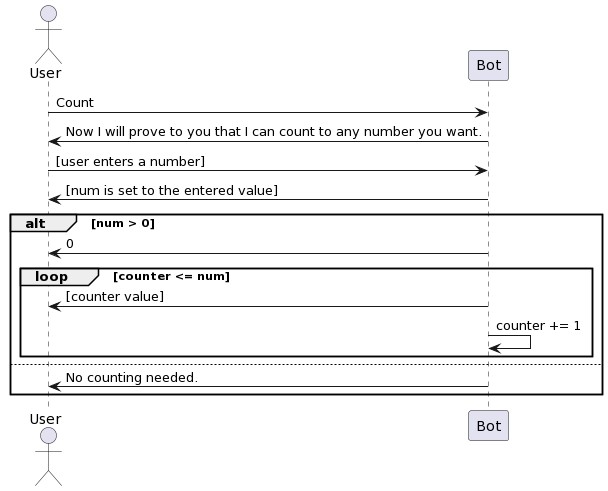
**ELEMENTS OF SEQUENCE DIAGRAM:**

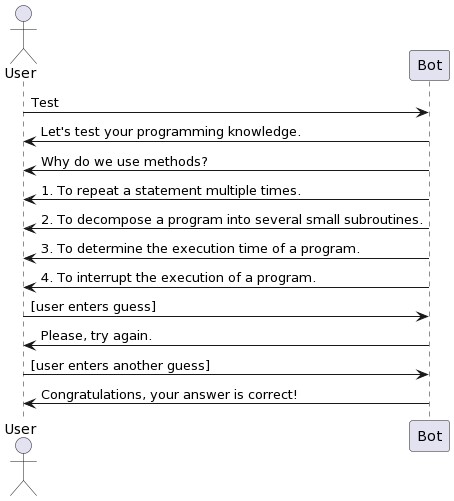
* + Objects
  + Links
  + Messages
  + Focus of control
  + Object life line

**SEQUENCE DIAGRAM**









#### 8.CONSTRUCTION OF COLLABORATION DIAGRAMS

Collaboration diagrams are the second kind of interaction diagram in the UML diagrams. They are used to represent the collaboration that realizes a use case. The most significant difference between the two types of interaction diagram is that a collaboration diagram explicitly shows the links between the objects that participate in a collaboration , as in sequence diagrams, there is no explicit time dimension.

**Message labels in collaboration diagrams:**

Messages on a collaboration diagram are represented by a set of symbols that are the same as those used in a sequence diagram, but with some additional elements to show sequencing and recurrence as these cannot be inferred from the structure of the diagram. Each message label includes the message signature and also a sequence number that reflects call nesting, iteration, branching, concurrency and synchronization within the interaction.

The formal message label syntax is as follows:

[predecessor] [guard-condition] sequence-expression [return-value ':='] message-name' (' [argument-list] ')'

**A *predecessor*** is a list of sequence numbers of the messages that must occur before the current message can be enabled. This permits the detailed specification of branching pathways. The message with the immediately preceding sequence number is assumed to be the predecessor by default, so if an interaction has no alternative pathways the predecessor list may be omitted without any ambiguity. The syntax for a predecessor is as follows:

sequence-number { ',' sequence-number} *'I'*

The *'I'* at the end of this expression indicates the end of the list and is only included when an explicit predecessor is shown.

***Guard conditions***are written in Object Constraint Language (OCL) ,and are only shown where the enabling of a message is subject to the defined condition. A guard condition may be used to represent the synchronization of different threads of control.

**A *sequence-expression***is a list of integers separated by dots ('.') optionally followed by a *name* (a single letter), optionally followed by a *recurrence* term and terminated by a colon. A sequence-expression has the following syntax:

integer { '.' integer } [name] [recurrence] ':'

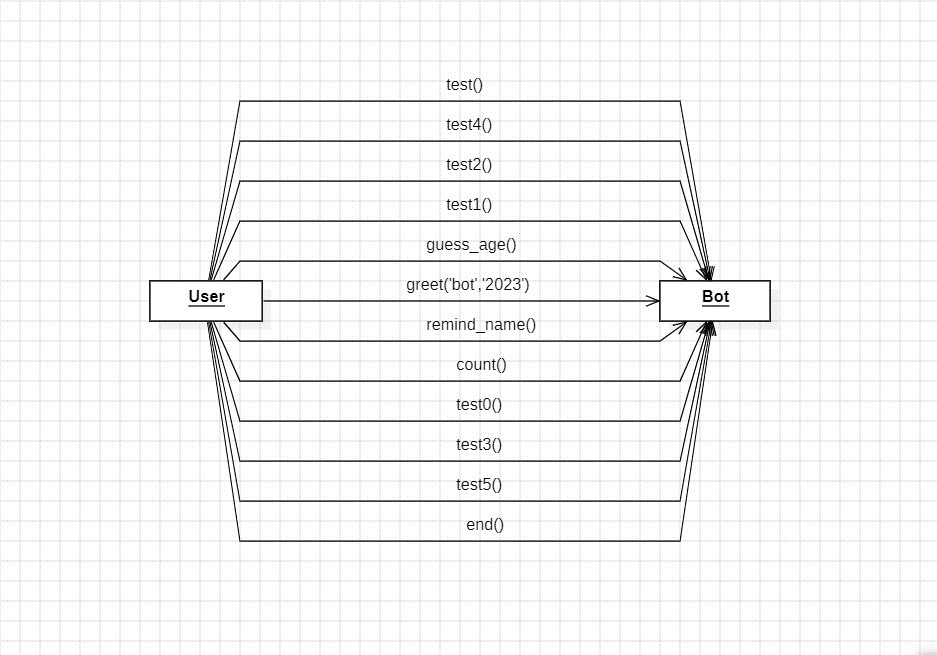
In this expression *integer* represents the sequential order of the message. This may be nested within a loop or a branch construct, so that, for example, message 5.1 occurs after message 5.2 and both are contained within the activation of message 5.

The *name* of a sequence-expression is used to differentiate two concurrent messages since these are given the same sequence number. For example, messages 3.2.1a and 3.2.1b are concurrent within the activation of message 3.2.

Recurrence reflects either iterative or conditional execution and its syntax is as follows:

*Branching:* '[ 'condition-clause‘ ] , *Iteration:* ‘ \* ‘ ‘ [ ‘ iteration-clause ‘ ] ' **Elements:**

 Objects ,Messages ,Path,Sequence Numbers,Links



#### 9.Construction of UML static class diagram

Class diagrams contain icons representing classes, packages, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model.

**Class:**

A Class a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.

Thus, a class is a template to create objects. Each object is an instance of

some class and objects cannot be instances of more than one class.

Classes should be named using the vocabulary of the domain.

For example, the Course Offering class may be defined with the following characteristics:

Attributes - location, time offered

Operations - retrieve location, retrieve time of day, add a student to the offering .

Each object would have a value for the attributes and access to the operations specified by the Course Offering class.

In the UML, classes are represented as compartmentalized rectangles.

The top compartment contains the name of the class.

The middle compartment contains the structure of the class (attributes).

The bottom compartment contains the behavior of the class (operations) as shown below.



**OBJECT :**

* AN OBJECT IS a representation of an entity, either real-world or conceptual.
* An object is a concept, abstraction, or thing with well defined boundaries and

meaning for an application.

* Each object in a system has three characteristics: state, behavior, and identity.

**STATE :** THE STATE OF an object is one of the possible conditions in which it may exist. The state of an object typically changes over time, and is defined by a set of properties (called attributes), with the values of the properties, plus the relationships the object may have with other objects.

For example, a course offering object in the registration system may be in one of two states: *open* and *closed.* It is available in the open state if value is < 10 otherwise closed.

**Behavior :**

* Behavior determines how an object responds to requests from other objects .
* Behavior is implemented by the set of operations for the object.

For example , In the registration system, a course offering could have the behaviors *add* a *student* and *delete* a *student.*

**Identity :**

* Identity means that each object is unique even if its state is identical to that of another object.

#### Attributes

Attributes are part of the essential description of a class. They belong to the class, unlike objects, which instantiate the class. Attributes are the common structure of what a member of the class can 'know'. Each object will have its own, possibly unique, value for each attribute.

Guidelines for identifying attributes of classes are as follows:

* Attributes usually correspond to nouns followed by prepositional phrases  Keep the class simple; state only enough attribute to defineobject state.
* Attributes are less likely to be fully described in the problem statement.
* Omit derived attributes.
* Do not carry discovery attributes to excess.

**STEREOTYPES AND CLASSES :**

As like stereotypes for relationships in use case diagrams. Classes can also have stereotypes. Here a stereotype provides the capability to create a new kind of modeling element. Here, we can create new kinds of classes. Some common stereotypes for a class are entity Class, boundary Class, control class, and exception.

**Entity Classes**

* An **entity class** models information and associated behavior that is generally long lived.
* This type of class may reflect a real-world entity or it may be needed to perform tasks internal to the system.
* They are typically independent of their surroundings; that is, they are not sensitive to how the surroundings communicate with the system.

**Boundary Classes :**

Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system (i.e., the interface to an actor). They constitute the surroundings dependent part of the system.

Boundary classes are used to model the system interfaces.

Boundary classes are also added to facilitate communication with other systems. During design phase, these classes are refined to take into consideration the chosen communication protocols.

**Control Classes**

* Control classes model sequencing behavior specific to one or more use cases.
* Control classes coordinate the events needed to realize the behavior specified in the use case.
* Control classes typically are application-dependent classes.

In the early stages of the Elaboration Phase, a control class is added

for each actor/use case pair. The control class is responsible for the flow of events in the use case.

In the early stages of the Elaboration Phase, a control class is added

for each actor/use case pair. The control class is responsible for the flow of events in the use case.

**NEED FOR RELATIONSHIPS AMONG CLASSES:**

All systems are made up of many classes and objects. System behaviour is achieved through the collaborations of the objects in the system.

Two types of relationships in CLASS diagram are:

1. Association Relationship 2. Aggregation Relationship

1. **Association Relationship:**

An association is a bidirectional semantic connection between classes. It is not a data flow as defined in structured analysis and design data may flow in either direction across the association. An association between classes means that there is a link between objects in the associated classes.

1. **Aggregation Relationship:**

An aggregation relationship is a specialized form of association in which a whole is related to its part(s). Aggregation is known as a “part-of” or containment relationship. The UML notation for an aggregation relationship is an association with a diamond next to the class denoting the aggregate(whole).

1. **Super-sub structure (Generalization Hierarchy):**

These allow objects to be build from other objects. The super-sub class hierarchy is a relationship between classes, where one class is the parent class of another class.

**NAMING RELATIONSHIP:**

An association may be named. Usually the name is an active verb or verb phrase that communicates the meaning of the relationship. Since the verb phrase typically implies a reading direction, it is desirable to name the association so it reads correctly from left to right or top to bottom. The words may have to be changed to read the association in the other direction (e.g., Buses are allotted to Routes). It is important to note that the name of the association is optional.

**ROLE NAMES:**

The end of an association where it connects to a class is called an association role.

Role names can be used instead of association names.

A role name is a noun that denotes how one class associates with another. The role name is placed on the association near the class that it modifies, and may be placed on one or both ends of an association line.

* It is not necessary to have both a role name and an association name.
* Associations are named or role names are used only when the names are needed for clarity.

**MULTIPLICITY INDICATORS:**

Although multiplicity is specified for classes, it defines the number of objects that participate in a relationship. Multiplicity defines the number of objects that are linked to one another. There are two multiplicity indicators for each association or aggregation one at each end of the line. Some common multiplicity indicators are

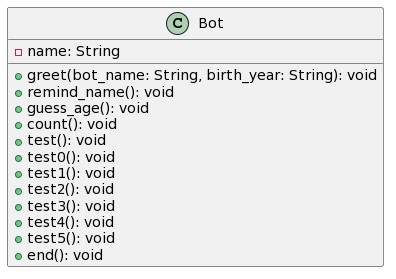
1 Exactly one

1. .. \* Zero or more
2. .. \* One or more

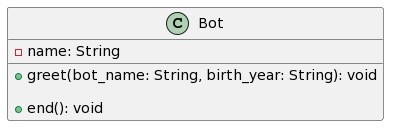
0. .. 1 Zero or one

5... 8 Specific range (5, 6, 7, or 8)

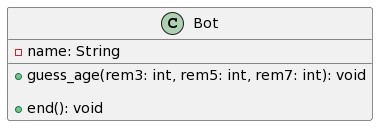
4... 7, 9 Combination (4, 5, 6, 7, or 9)



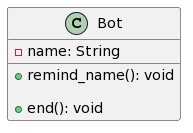
**CLASS DIAGRAM FOR GREETING**



**CLASS DIAGRAM FOR GUESSING AGE**



**CLASS DIAGRAM FOR USER NAME REMINDER**



#### 10.Analyzing the object behavior by constructing the UML State Chart diagram

Use cases and scenarios provide a way to describe system behavior; in the form of interaction between objects in the system. Sometimes it is necessary to consider inside behavior of an object.

A state chart diagram shows the **states** of a single object, the events or messages that cause a **transition** from one state to another, and the **actions** that result from a state change. As in Activity diagram , state chart diagram also contains special symbols for start state and stop state.

State chart diagram cannot be created for every class in the system , it is only for those class objects with significant behavior.

State chart diagrams are closely related to activity diagrams. The main difference between the two diagrams is state chart diagrams are state centric, while activity diagrams are activity centric. A state chart diagram is typically used to model the discrete stages of an object’s lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process.

**STATE:**

A state represents a condition or situation during the life of an object during which it satisfies some condition, performs some action or waits for some event.

UML notation for STATE is

To identify the states for an object its better to concentrate on sequence diagram. In an ESU the object for Course Offering may have in the following states, initialization, open and closed state. These states are obtained from the attribute and links defined for the object. Each state also contains a compartment for actions.

**Actions:**

Actions on states can occur at one of four times:

* on entry
* on exit
* do
* on event.

**on entry :**What type of action that object has to perform after entering into the state. **on exit :** What type of action that object has to perform after exiting from the state. **Do :**The task to be performed when object is in this state, and must to continue until it leaves the state.

**on event :** An on event action is similar to a state transition label with the following syntax: event(args)[condition] : the Action **State Transition:**

A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state. You can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

**Transitions are labeled with the following syntax:** event (arguments) [condition] / action ^ target. send Event (arguments) Only one event is allowed per transition, and one action per event.

State Details :

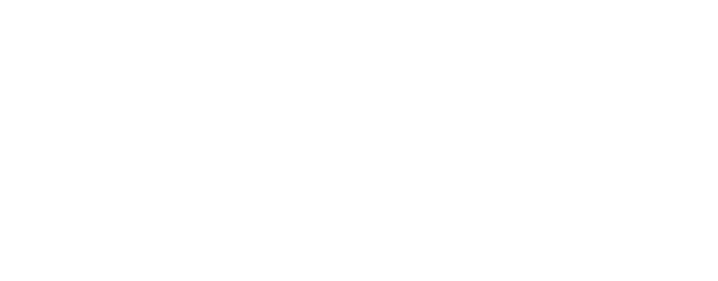
Actions that accompany all state transitions into a state may be placed as an entry action within the state. Like wise that accompany all state transitions out of a state may be placed as exit actions within the state. Behavior that occurs within the state is called an activity.

An activity starts when the state is entered and either completes or is interrupted by an outgoing state transition. The behavior may be a simple action or it may be an event sent to another object.

UML notation for State Details:

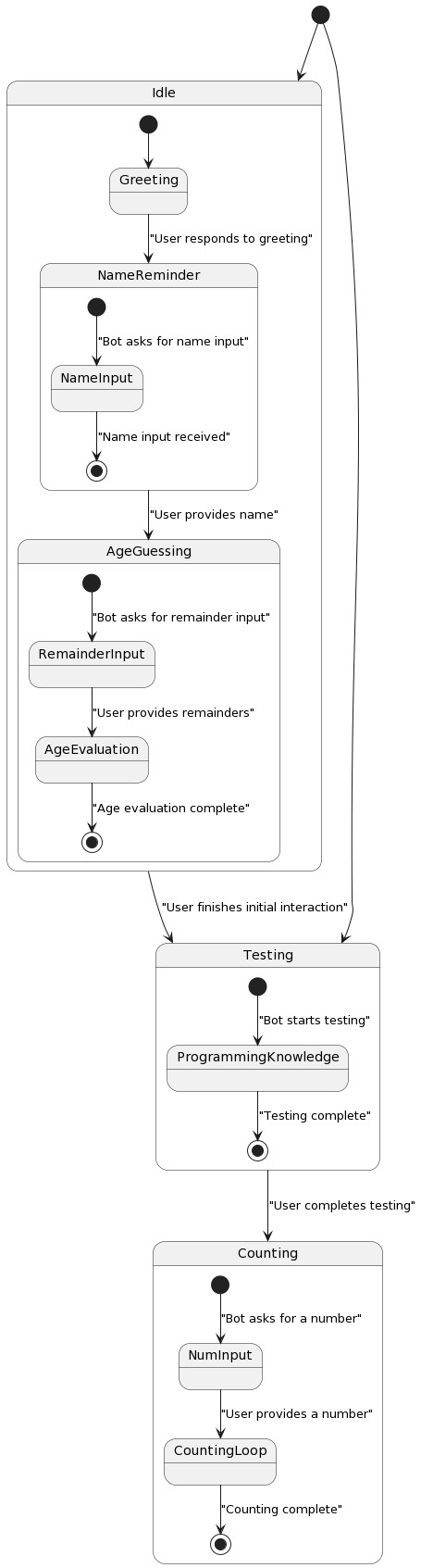
##### StateName

entry/ simple action entry/ ^class name.eventname do/ simple action do/ ^class name.event name exit/ ^class name.event name



Purpose of State chart diagram:

* State chart diagrams are used to model dynamic view of a system.
* State chart diagrams are used to modelling lifetime of an object.
* State chart diagrams are used to focus on the changing state of a system driven by events.
* It will also be used when showing the behavior of a class over several use cases.



#### 11.CONSTRUCTION OF IMPLEMENTATION DIAGRAMS

**Component diagrams:**

In a large project there will be many files that make up the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compiletime, at link-time or at run-time. There are also dependencies between source code files and the executable files or byte code files that are derived from them by compilation. Component diagrams are one of the two types of implementation diagram in UML. Component diagrams show these dependencies between software components in the system. Stereotypes can be used to show dependencies that are specific to particular languages also.

A component diagram shows the allocation of classes and objects to components in the physical design of a system. A component diagram may represent all or part of the component architecture of a system along with dependency relationships.

**COMPONENT DIAGRAM FOR FRAMS**

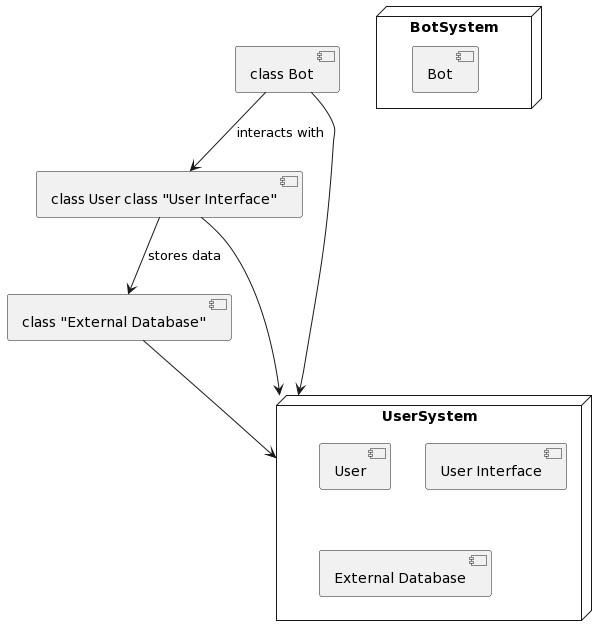


**Deployment diagrams:**

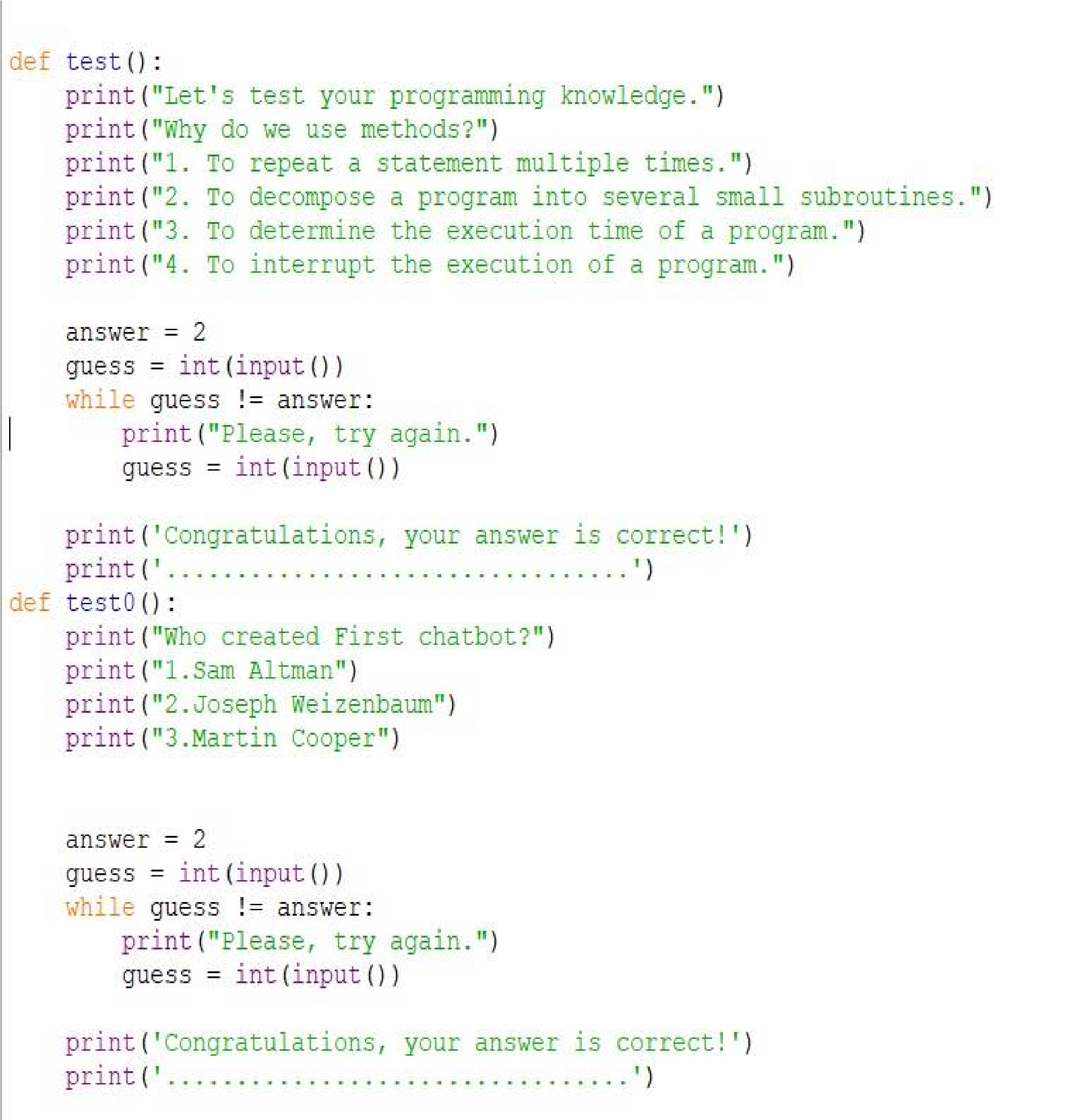
The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of run-time processing elements and the software components and processes that are located on them.

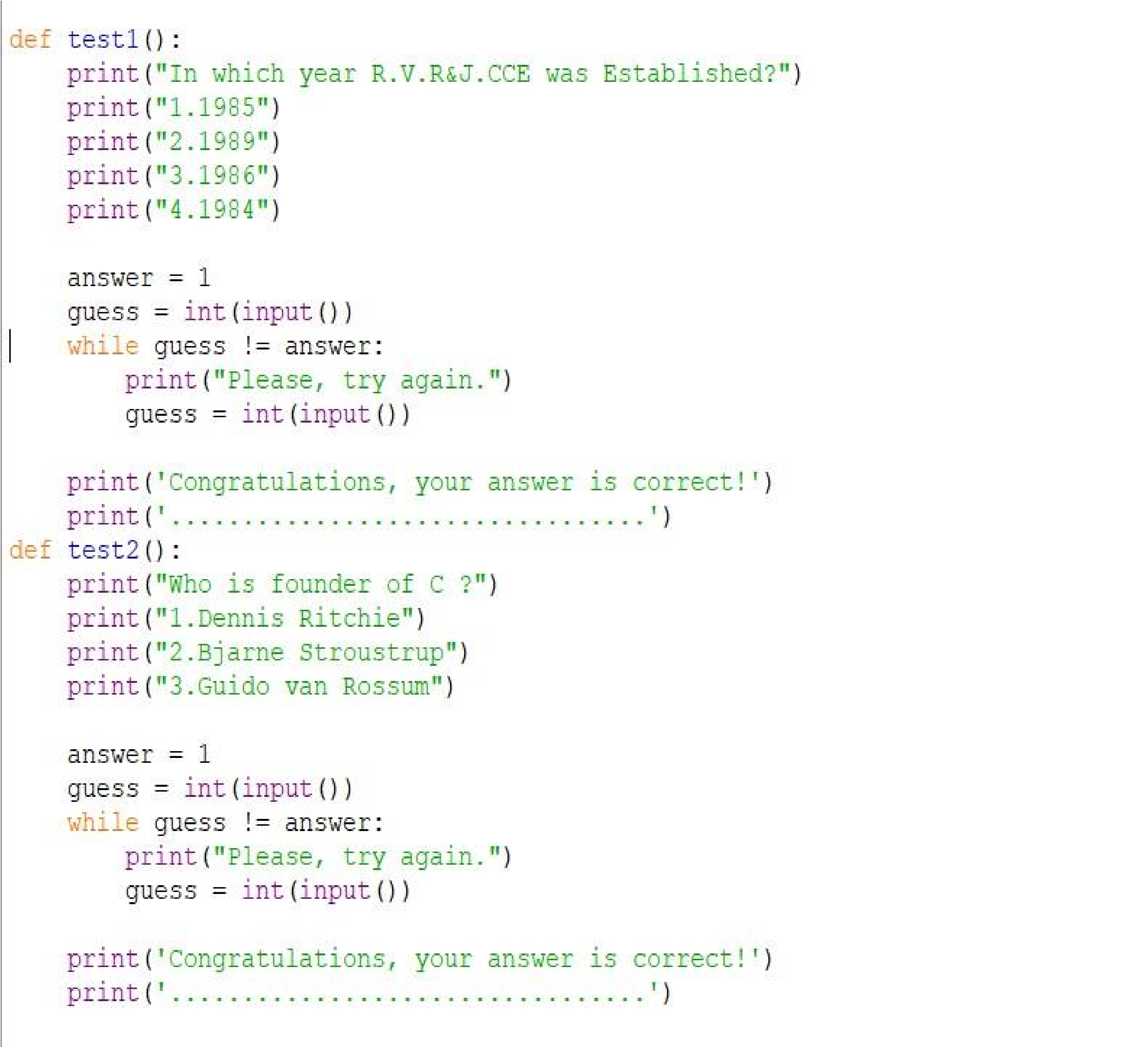
Deployment diagrams are made up of nodes and communication associations. Nodes are typically used to show computers and the communication associations show the network and protocols that are used to communicate between nodes. Nodes can be used to show other processing resources such as people or mechanical resources.

Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes connected by communication associa

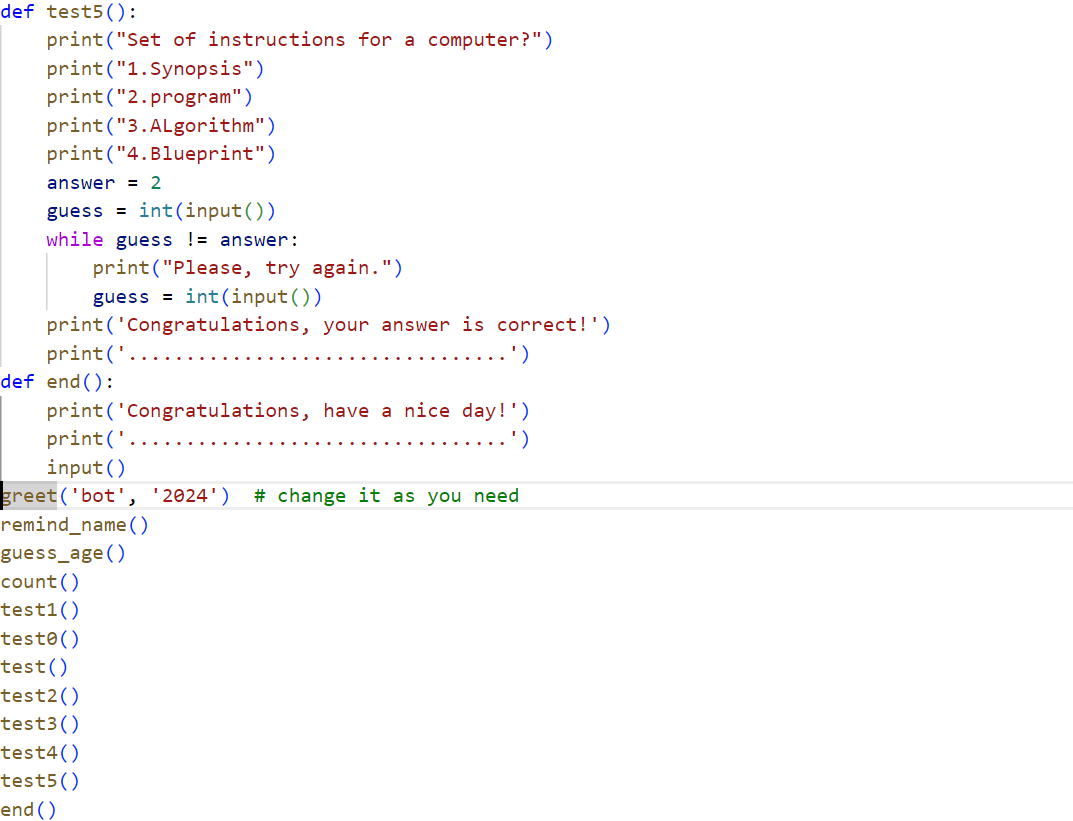


##### 12. SAMPLE APPLICATION CODE

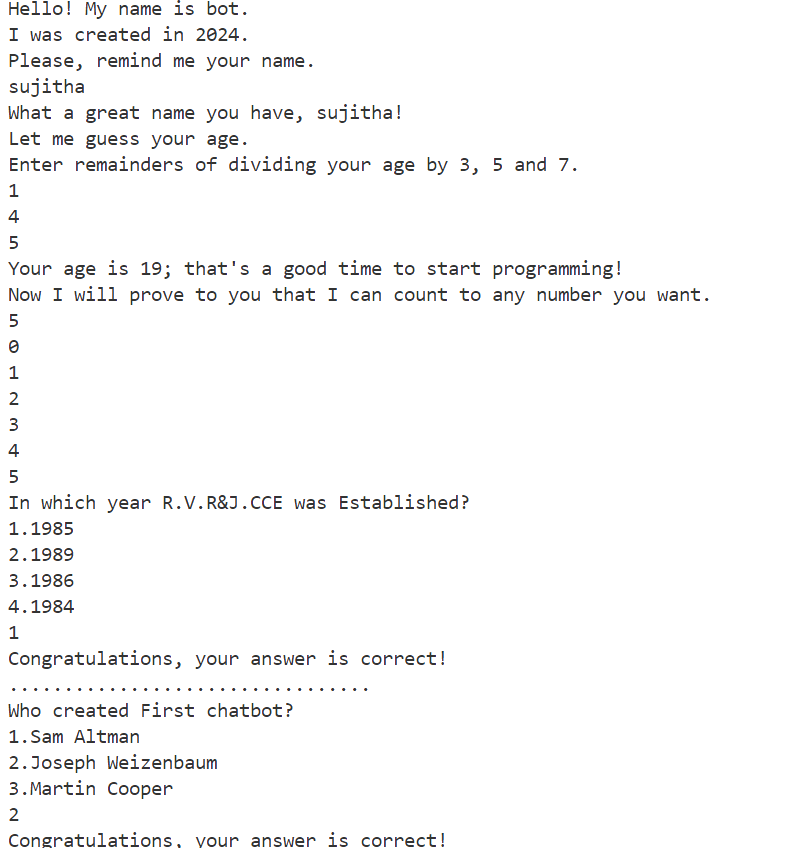


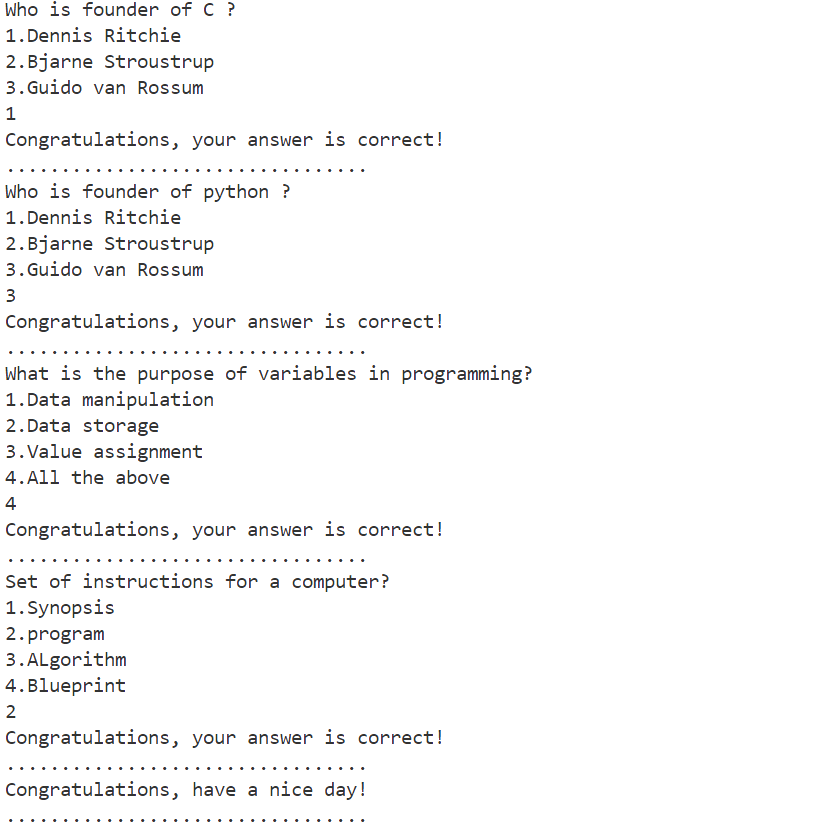






OUTPUT SCREENSHOTS :





#### 13.TESTING

The main purpose of testing FRAMS is to ensure that all the activities and functionalities of this software run smoothly with no errors and it remains protected.

FOR ADMIN :

* Verify Admin only can do chat with chatbot
* Verify Admin Can’t update information
* Verify Admin Can’t view information
* Verify Admin Can’t delete information
* Verify Admin Can’t add information

* FOR USER :
  + Verify User can do chat with chatbot
  + Verify User can update information
  + Verify User can view information
  + Verify User can delete information
  + Verify User can add information

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case no** | **Functionality to be Tested** | **Actual input** | **Actual output** | **Expected output** | **Status** |
| 1 | Remind name | User name | Wishes the User | Wishes the User | Pass |
| 2 | Guess the age | Remainders of dividing user age by 3, 5 and 7 | User age | User age | Pass |
| 3 | Count | Any number to be counted | Counted Number | Counted Number | Pass |
| 4 | Testing | Answer | Congratulations your answer is  correct | Congratulations your answer is  correct | Pass |

## 14.CONCLUSION

When testing the last prototype we got findings suggesting that the participants did not have a problem with getting information from a chatbot instead of a human. The information that they got was not seen as less trustworthy, this could be supported by the fact that the chatbot provided a source for the information it gave. It has been interesting to investigate how the participants interacted with the chatbot and how they reported on it afterwards. Our findings have some indicators leading towards that a chatbot could be a good alternative for acting as a helpful friend for freshmans at a new school. Still we have to stress the fact that the chatbot was not very intelligent and that the evaluators had to adjust their language to match the chatbots.

### REFERENCES

1.Prompt Engineering Bible,Robert E. Miller

2.Chatbots: An Introduction And Easy Guide To Making Your Own Oisin Muldowney