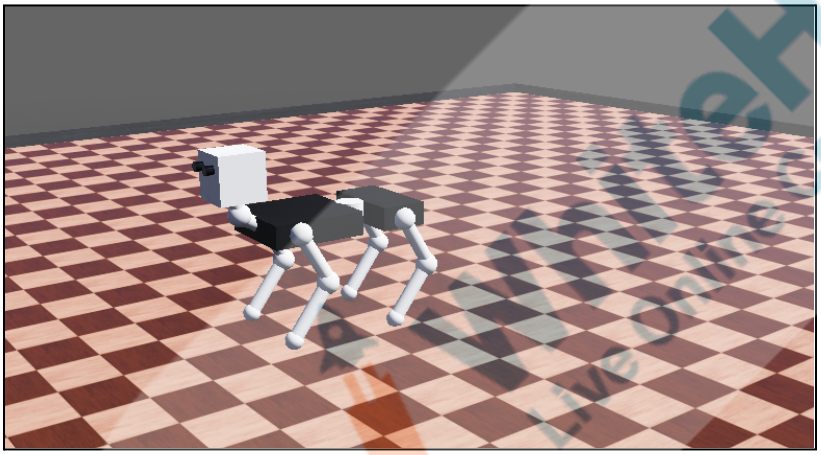
Topic	FOUR LEGGED ROBOT - III	
Class Description	**Students will add a human robot to our four-legged robot's environment. They will also learn about emitter and receiver nodes. Additionally, they will implement the code to move the dog according to the human robot's command.**	
Class	**PRO C294**	
Class time	**45 mins**	
Goal	• Adding a pedestrian robot to the environment • Sending messages using emitter • Receiving messages using receiver	

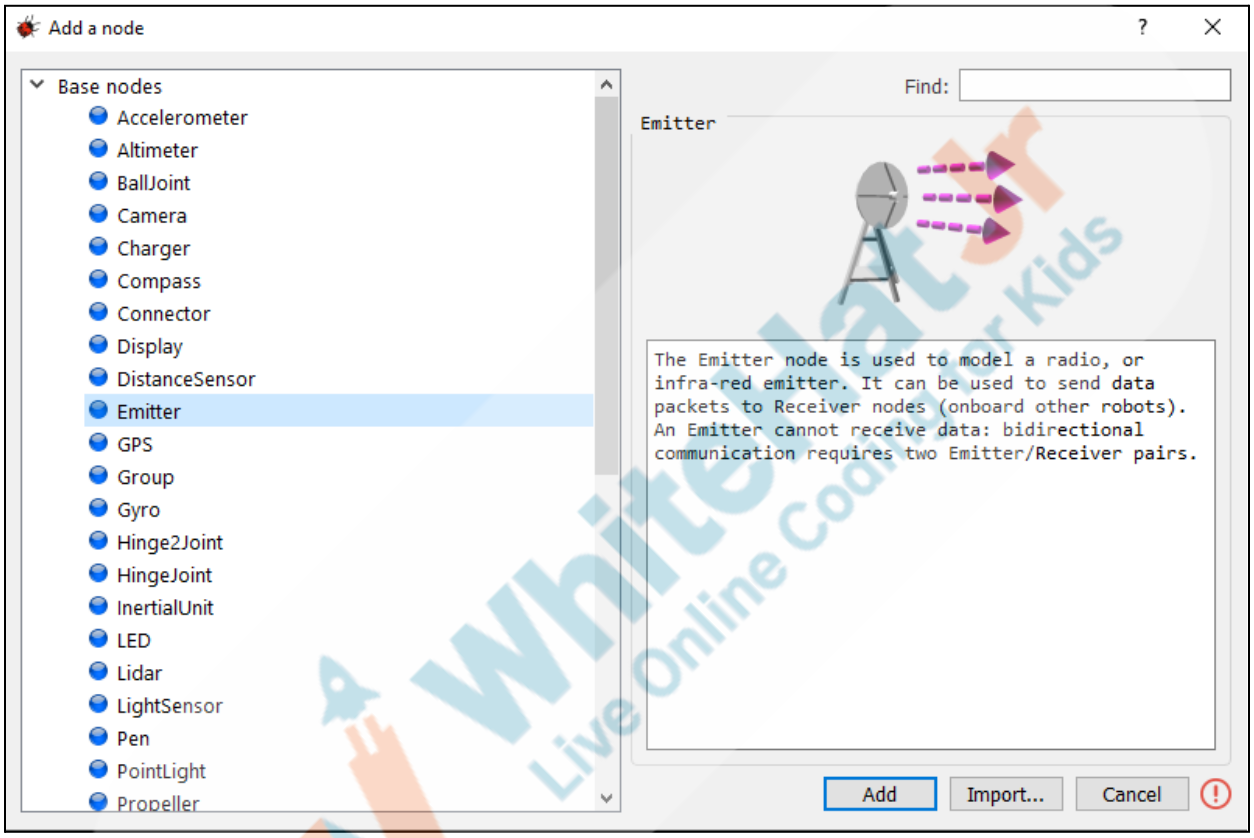
Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Smartphone
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen	

Class structure	**Warm-Up** **Teacher-Led Activity 1** **Student-Led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit & Permissions:	**This project uses Webots, an open-source mobile robot simulation software developed by Cyberbotics Ltd.** **License**	
WARM-UP SESSION - 10 mins		

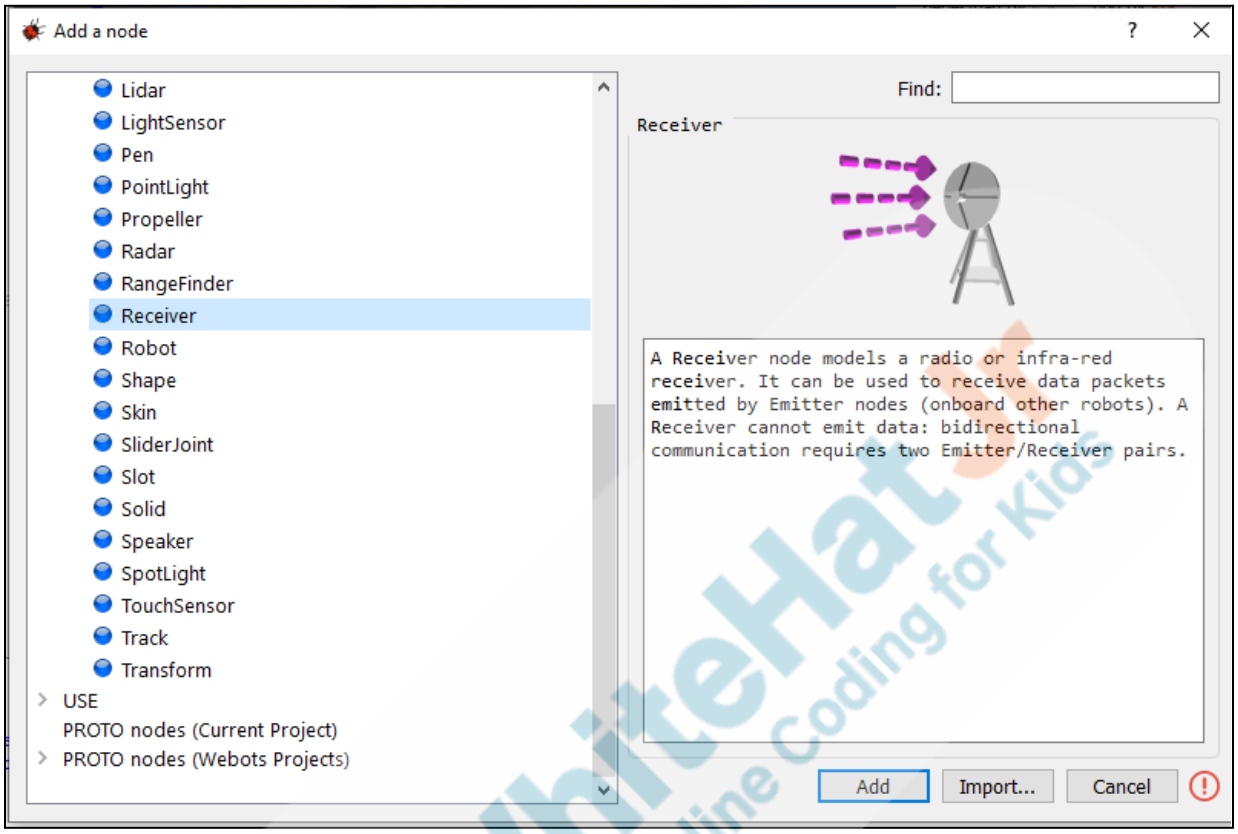
Teacher Action	Student Action
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
• **Adding the human robot** • **Adding Emitter node to the human robot** • **Coding the emitter node to send data**	
Teacher Action	**Student Action**
So what did we learn in our last class? *The teacher will clarify if there are any doubts!* So let's get started with today's class. In our last class, we have written the controller to make the Four-legged robot walk.	**ESR:** Varied!

Today we will program it to follow commands.	
Teacher downloads the boilerplate code from Teacher Activity 1.	
Open the file using Webot. 1. Open the **webots** 2. Go to the **Open World** 3. Upload the webots file from boilerplate code	
Teacher opens the boilerplate code on webots.  *Pause* ⏸ *the simulation before you start coding it.* We had completed the controller code in the last class. Today, we want to add more elements to our project. We will add a predefined human robot to our project. The human will give commands to the dog robot and the dog robot will act accordingly. How do you think we can do that?	**ESR:** We need to use some special node which can

Exactly! This special node is called an **Emitter**. An emitter is a device which can send data over a channel. It can be a radio or an infrared emitter.	send data.



This data can be received by a **Receiver**. An **Emitter** can only send the data and a **Receiver** can only receive the data. This communication cannot be bidirectional.

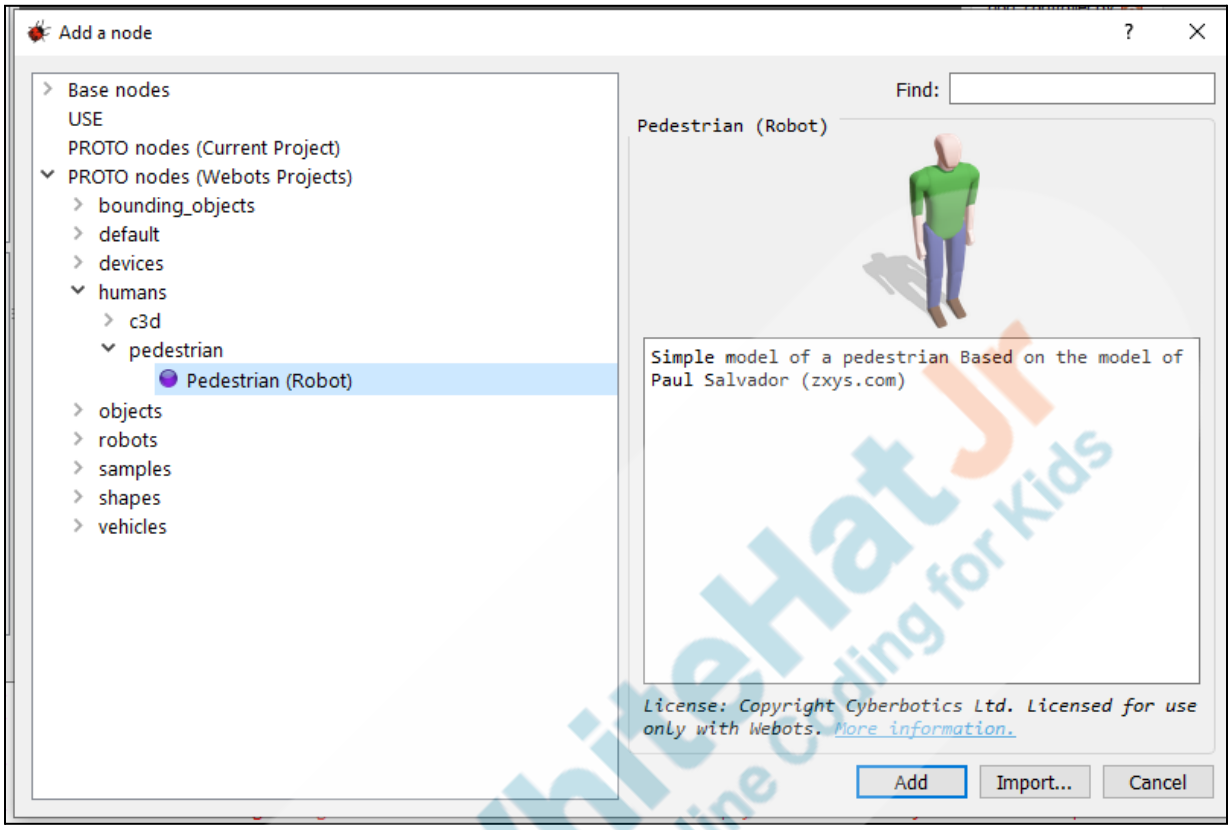


Also, an **Emitter** can send data to the **Receiver** of another robot. It cannot send data to the receiver of the same robot.

Note: Make sure you save the project after every major step. Otherwise, you might lose your work.

1. Let's add the human robot to our environment first.

   Click on the **Scene Panel** → click on the **Add New** button  → go to **PROTO nodes** → click on **humans** → select **Pedestrian**

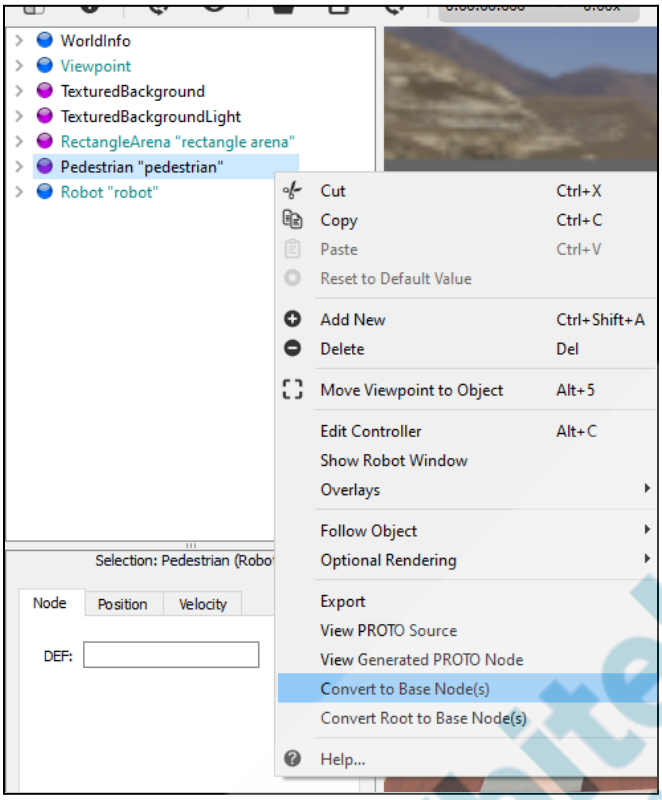


2. Now, what should we do?

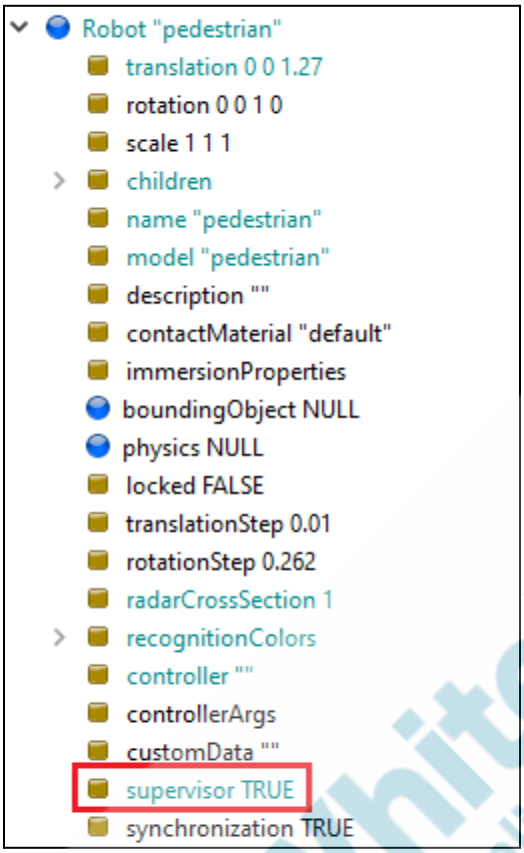
   Exactly.We have to add the **Emitter** device to the **Pedestrian** robot.

   a. Right-click on the Pedestrian robot in the **Scene Panel** and select **convert to base nodes.** By doing this, we can view the base nodes for a pre-built robot.

ESR: We need to add the **Emitter** node to the **Pedestrian** robot.

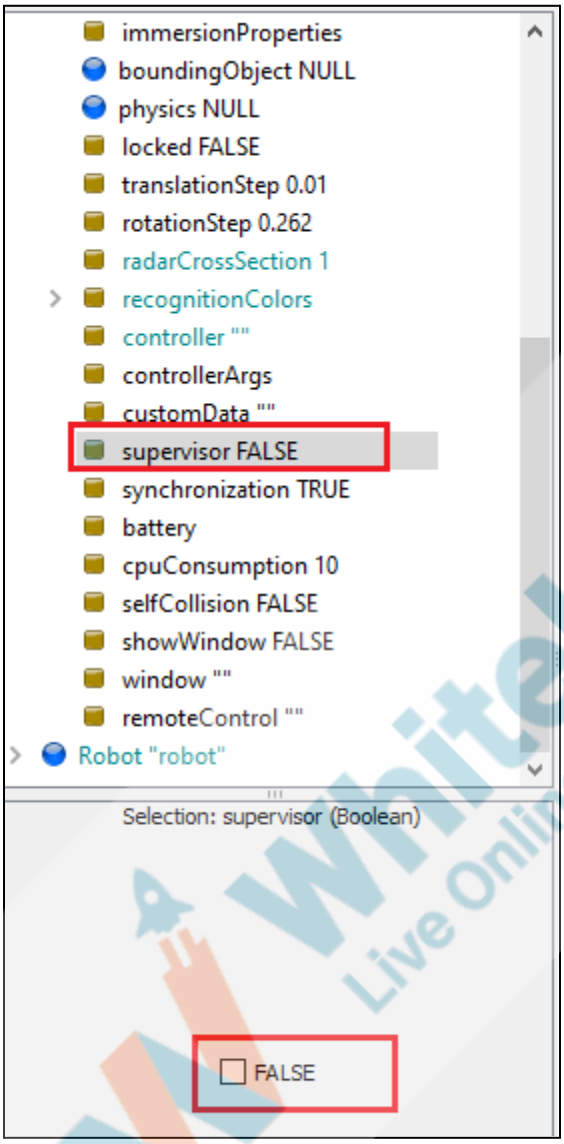


b. Now, we can see the **base nodes** of the pedestrian robot. Here, you will notice one property named **Supervisor.**

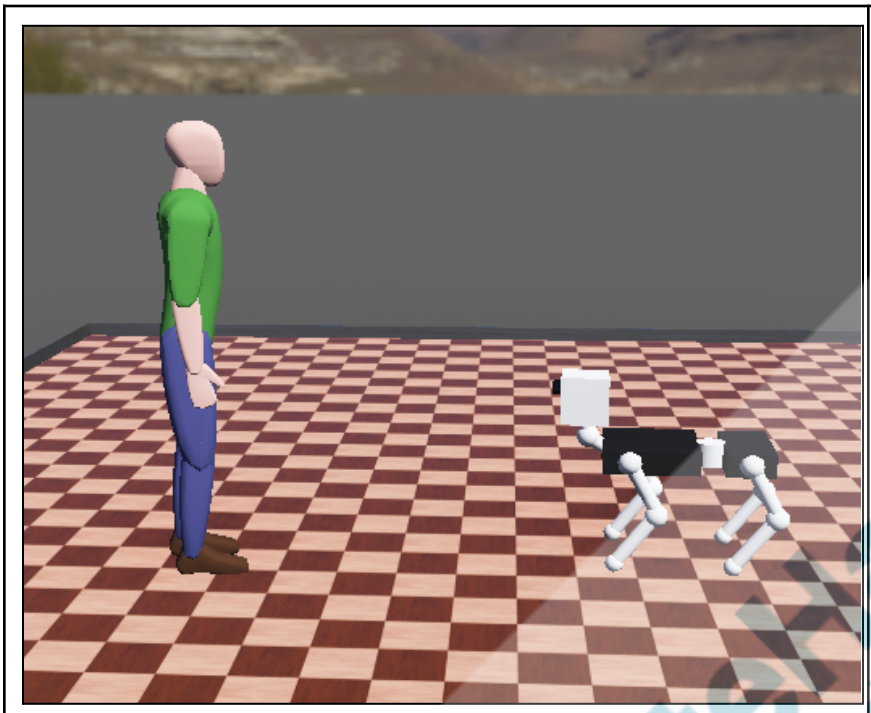


A **Supervisor** is a special type of robot which has additional powers. We do not require a **Supervisor** robot in this project. So, we will change it to a regular **Robot,** instead of a **Supervisor**.

The **Supervisor** property is set as **TRUE** for the **Pedestrian** robot. We will change it to **FALSE**.

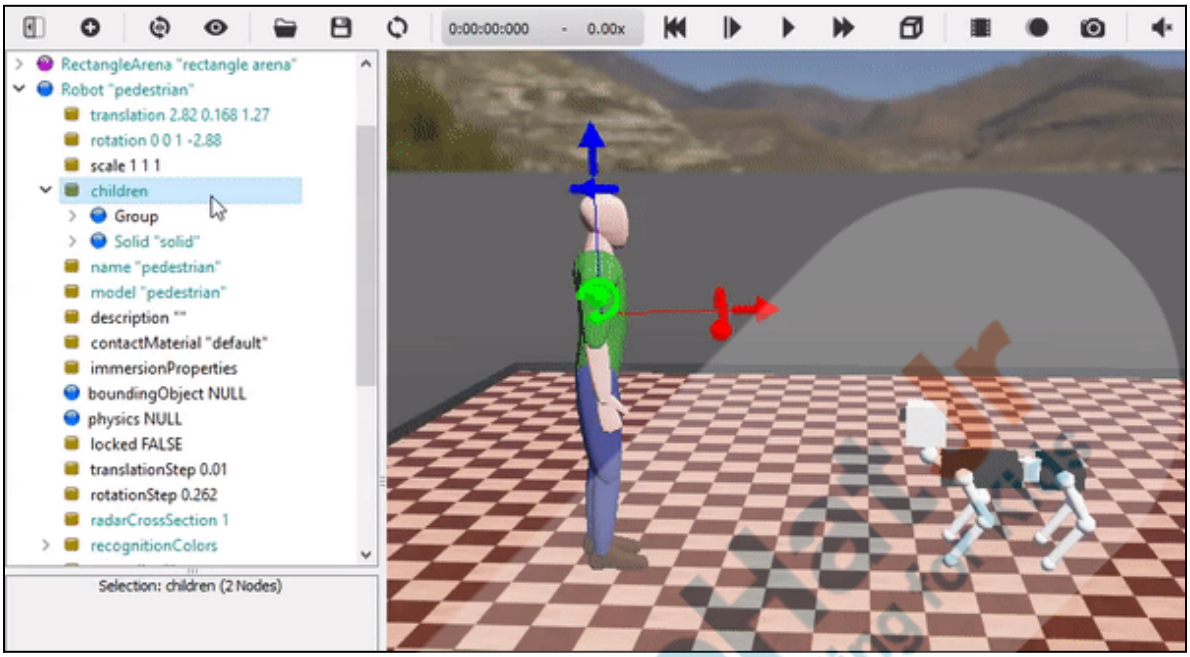


c. Rotate and position the **Pedestrian** robot in front of the dog robot.



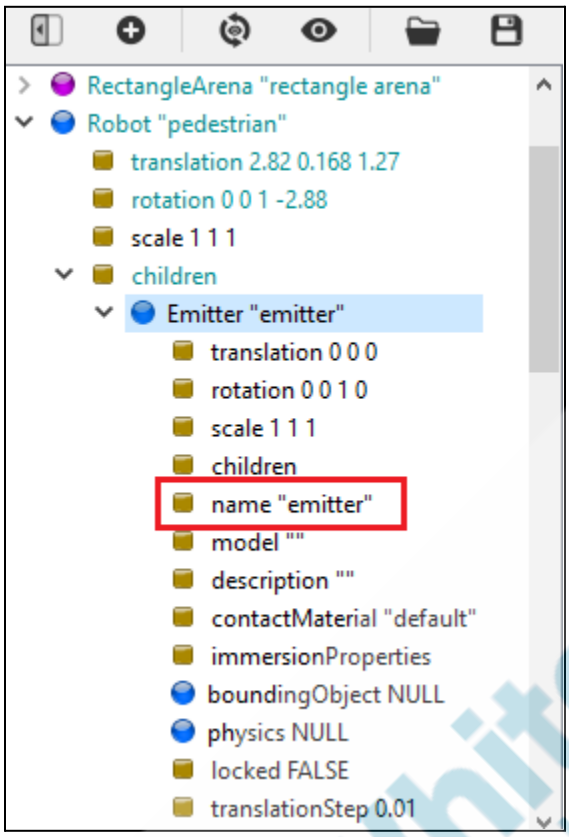
d. Let's add the **Emitter** node in the **Pedestrian** robot now.

   Go to **Pedestrian** robot→ right-click on the **children** property → click on **Add New** → select **Emitter** under **Base nodes** → click on **Add**



Click here to view the reference video.

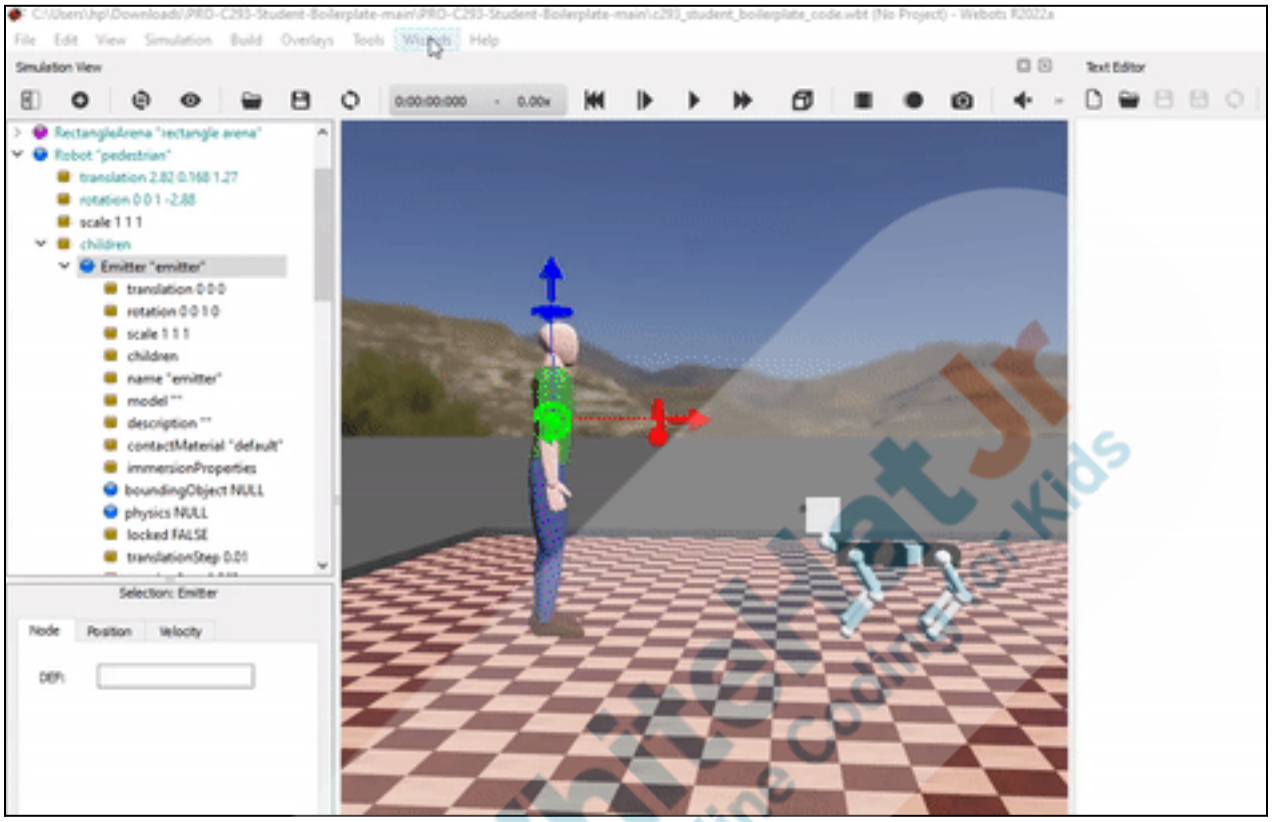
Notice that the name of the **Emitter** device is **emitter.** We will use this name in the controller to program the **Emitter** device.	



Code:

Let's code the **Emitter** device now. To send "sit", "stand" and "walk" commands.

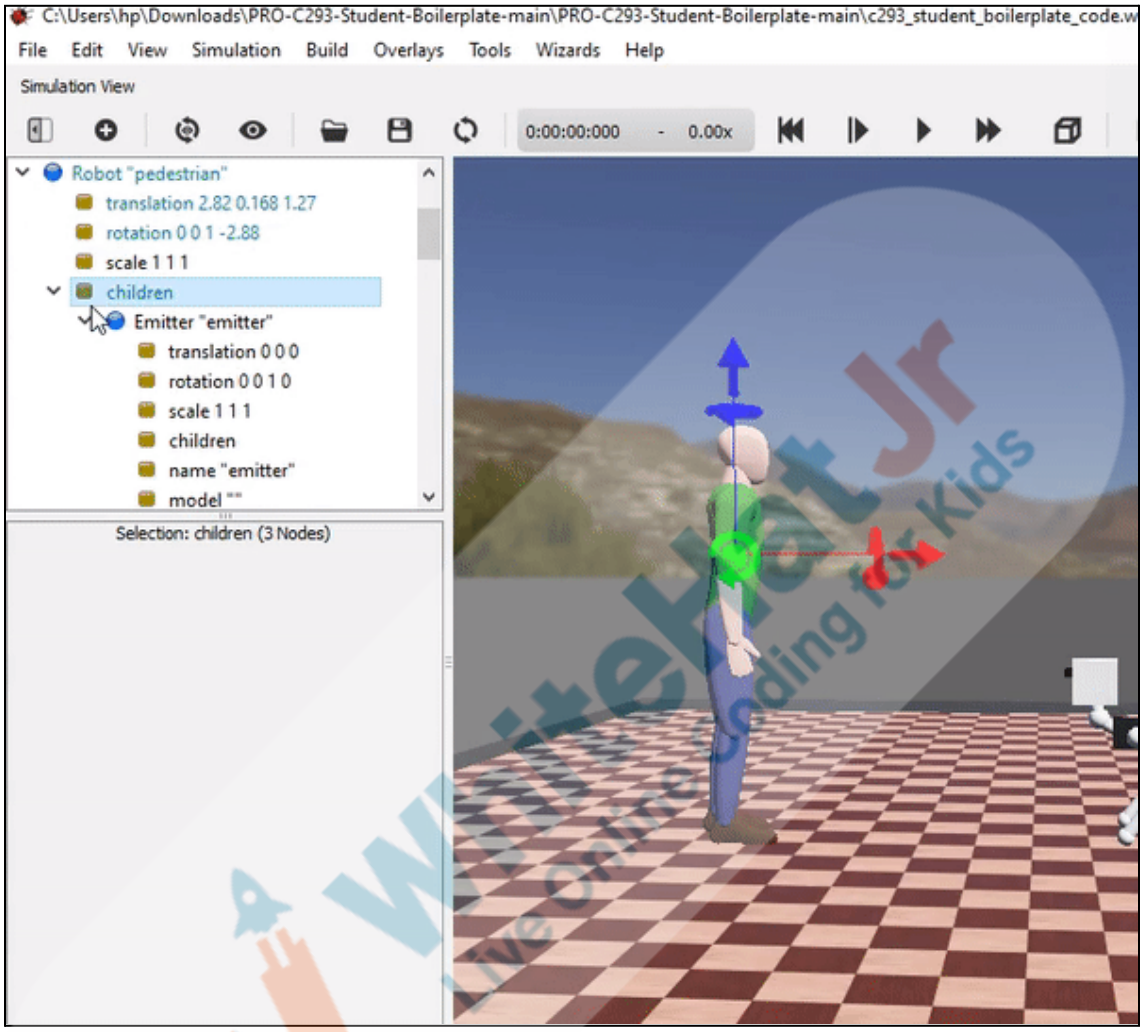
1. Let's create a new controller for the **Pedestrian** robot.
   Go to wizards in the **menu bar** → click on the **New Robot Controller** → create a new python controller → name it as **human.py**



[Click here](#) to view the reference video.

2. Change the controller in the **Pedestrian** robot.

   Go to **Pedestrian** robot in the **Scene Panel** → go to **controller** property → select the newly created **human** controller.



Click here to view the reference video.

Let's write the code now.

1. What should be the first line of code?

   Exactly. Let's do that.

   from controller import Robot

2. Let's create an instance of the robot. Aslo, initiate

ESR: We need to import **Robot** from the controller.

the **timestep** variable at 64

```
robot = Robot()

timestep = 64
```

3. Create a reference for the **emitter** device. We will do it using the same **getDevice()** method that we used for the motors.

```
emitter=robot.getDevice("emitter")
```

4. We need to add some delay after the **emitter** sends each message. We can do it by writing a method. We will name it as the **add_delay()** method.

   We will pass a parameter named **num_timestep.** This will specify for how many timesteps we want to make the program wait.

   We will initiate a **counter** named **time_counter** at 0. A loop will run until the **time_counter** is the same as **num_timestep**. This will make code wait.

```
def add_delay(num_timestep):
    time_counter = 0
    while robot.step(timestep)  !=  -1:
        if time_counter >= num_timestep:
            break
        time_counter += 1
```

5. Let's initiate an array named **commands** where we will store different commands for the dog robot.

```
commands = ["sit","stand","walk","stand"]
```

6. Now, how to send the data?

ESR: There must be a method to send data from the emitter.

Right! We have a **send()** method to send the data from the emitter.

But this method can only send byte messages.

So, if we want to send a string “hello”, we have to convert it to a byte message first and then send it.

Example-

info = bytes(‘hello’, 'utf-8')

emitter.send(info)

7. As we have the **commands** array which we want to use to send a message, we will use **while** loop to send each command one by one.

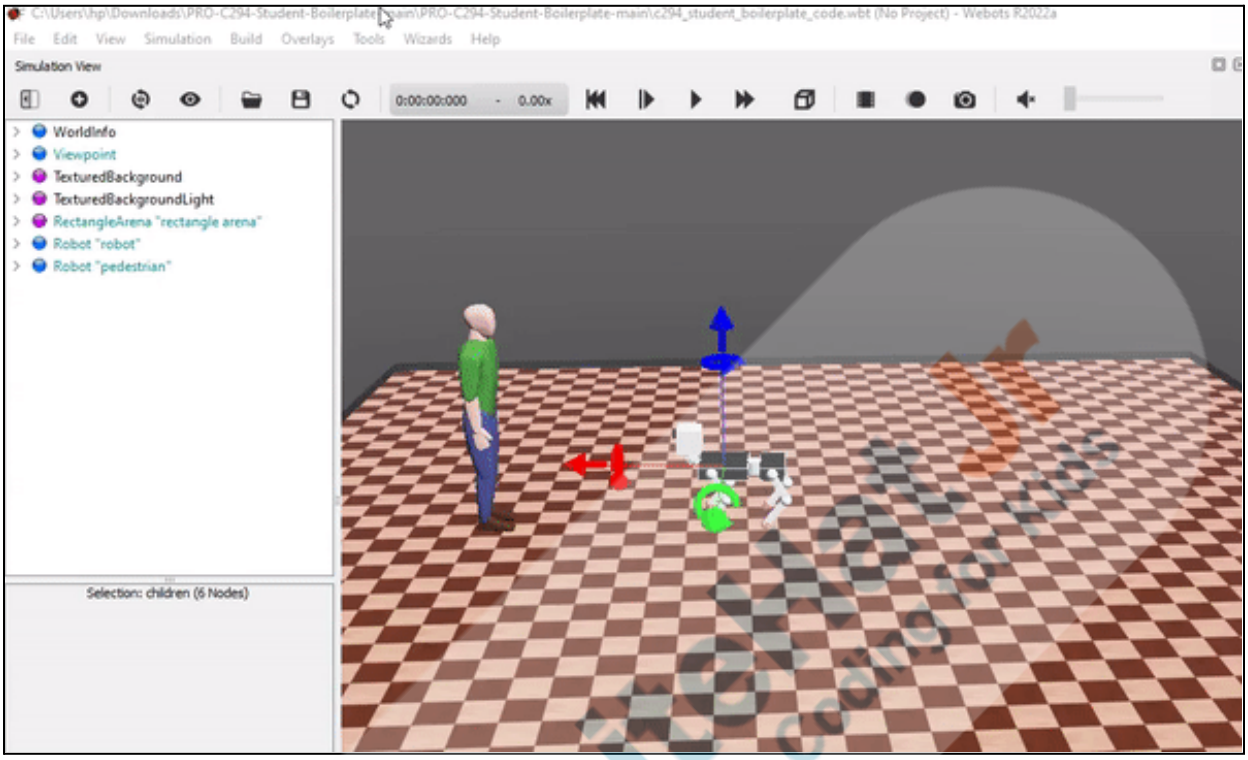
```
robot = Robot()
timestep = 64
emitter=robot.getDevice("emitter")

def add_delay(num_timestep):
    time_counter = 0
    while robot.step(timestep)  !=  -1:
        if time_counter >= num_timestep:
            break
        time_counter += 1


commands=["sit","stand","walk","stand"]

i=0

while robot.step(timestep) != -1:

    while(i<20):
        info = bytes(commands[i%4], 'utf-8')
        emitter.send(info)
        add_delay(50)
        i=i+1
```

Reference code:

```
from controller import Robot


robot = Robot()
timestep = 64
emitter=robot.getDevice("emitter")


def add_delay(num_timestep):

    time_counter = 0
    while robot.step(timestep)  !=  -1:
        if time_counter >= num_timestep:
            break
        time_counter += 1


commands=["sit","stand","walk","stand"]

i=0



while robot.step(timestep) != -1:

    while(i<20):
        info = bytes(commands[i%4], 'utf-8')
        emitter.send(info)
        add_delay(50)
        i=i+1
```

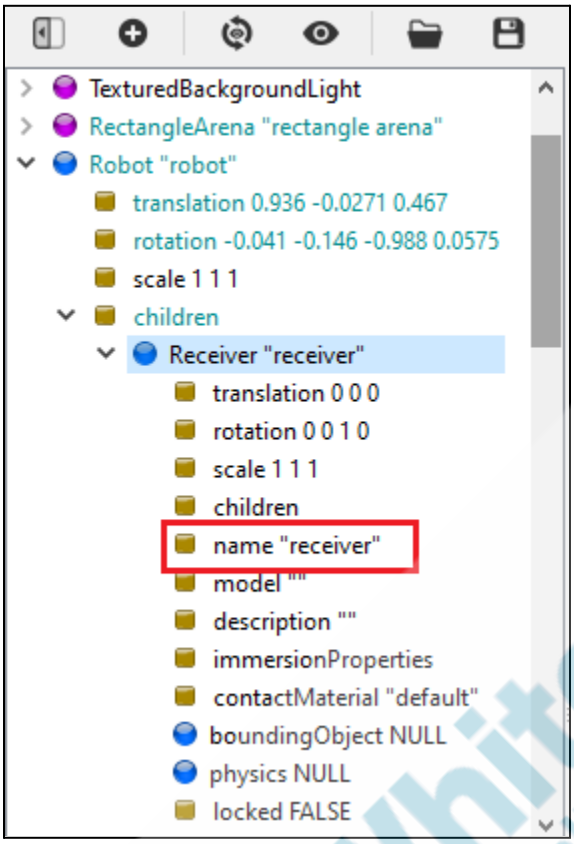
Looks good, isn't it? What's left?	**ESR:** We need to add the receiver to the **dog** robot.
Yes. We need to add the receiver to the **dog** robot. Let's get started.	
Are you ready?	**ESR:** Yes.

Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
STUDENT-LED ACTIVITY - 20 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Full Screen.**	
Student Initiates Screen Share	
ACTIVITY • **Learn to add a receiver node.** • **Code the receiver node to receive messages from the emitter.** • **Move the dog according to the human's instructions.**	
Teacher Action	**Student Action**
Student downloads the boilerplate code from Student Activity 1.	
Let's add the **Receiver** node to the **dog** robot. 1. Right click on the children node of the dog robot → click on **Add New** → go to **Base Nodes** → select **Receiver**.	



Click here to view the reference video.

2. Observe that the name of the **Receiver** node is set as **receiver.** We will use this name to write the program for our **controller.**



3. Let's write the controller now.

   We already have the code to make the robot walk. Let's write it as a separate function first.

dog_controller.py* human.py

```
from controller import Robot

robot=Robot()

timestep=320

flag= 0

leg1=robot.getDevice("front_left")
leg2=robot.getDevice("front_right")
leg3=robot.getDevice("back_left")
leg4=robot.getDevice("back_right")

def walk(flag):
    if(flag%10==0):
        leg1.setPosition(-0.3)
    elif(flag%10==2):
        leg2.setPosition(-0.3)
    elif(flag%10==4):
        leg3.setPosition(-0.3)
    elif(flag%10==6):
        leg4.setPosition(-0.3)
    elif(flag%10==7):
        leg1.setPosition(0.2)
        leg2.setPosition(0.2)
        leg3.setPosition(0.2)
        leg4.setPosition(0.2)

while (robot.step(timestep) !=-1):

    flag=flag+1
```

4. Let's write the **add_delay()** method as well.

```
def add_delay(num_timestep):
    time_counter = 0
    while robot.step(timestep)  !=  -1:
        if time_counter >= num_timestep:
            break
        time_counter += 1
```

5. Let's write a method which will make the dog sit.

```
def sit():
    leg1.setPosition(0)
    leg2.setPosition(0)
    leg3.setPosition(-0.37)
    leg4.setPosition(-0.37)
    add_delay(3)
```

6. Let's write a method to make the dog stand.

```
def stand():
    leg1.setPosition(0)
    leg2.setPosition(0)
    leg3.setPosition(0)
    leg4.setPosition(0)
    add_delay(3)
```

7. We have the method to add motion to the dog now. Let's add the receiver device in the code and enable it.

```
from controller import Robot

robot=Robot()
timestep=64

leg1=robot.getDevice("front_left")
leg2=robot.getDevice("front_right")
leg3=robot.getDevice("back_left")
leg4=robot.getDevice("back_right")

receiver= robot.getDevice("receiver")
receiver.enable(timestep)

```

8. Initiate a variable named **received_data** at the top.

   received_data=0

9. Now, let's go to the main **while** loop and let's write the code to fetch commands from the **Emitter**.

   a. We have a method named **getData()** for the receiver devices which we will use to fetch the data from the **Emitter**.
   b. But we need to make sure that the **getData()** method is called only when the receiver queue is not empty. We will use the **getQueueLength()** method for that.
   c. Once we receive the data, we will print it and we will ask the receiver to shift to the next packet by using the **nextPacket()** method.

```
while (robot.step(timestep) !=-1):

    if receiver.getQueueLength() > 0:
        received_data = receiver.getData().decode()
        print(received_data, type(received_data))
        receiver.nextPacket()
```

10. Now, depending on the value of the **received_data** variable, we will call the methods- **sit(), stand()** and

walk().

```
if received_data == "sit":
    sit()
    received_data=""

elif received_data == "stand":
    stand()
    received_data=""

elif received_data == "walk":
    flag = flag + 1
    while(flag%30 != 0):
        walk(flag)
        flag=flag+1
        add_delay(3)
    received_data=""
```

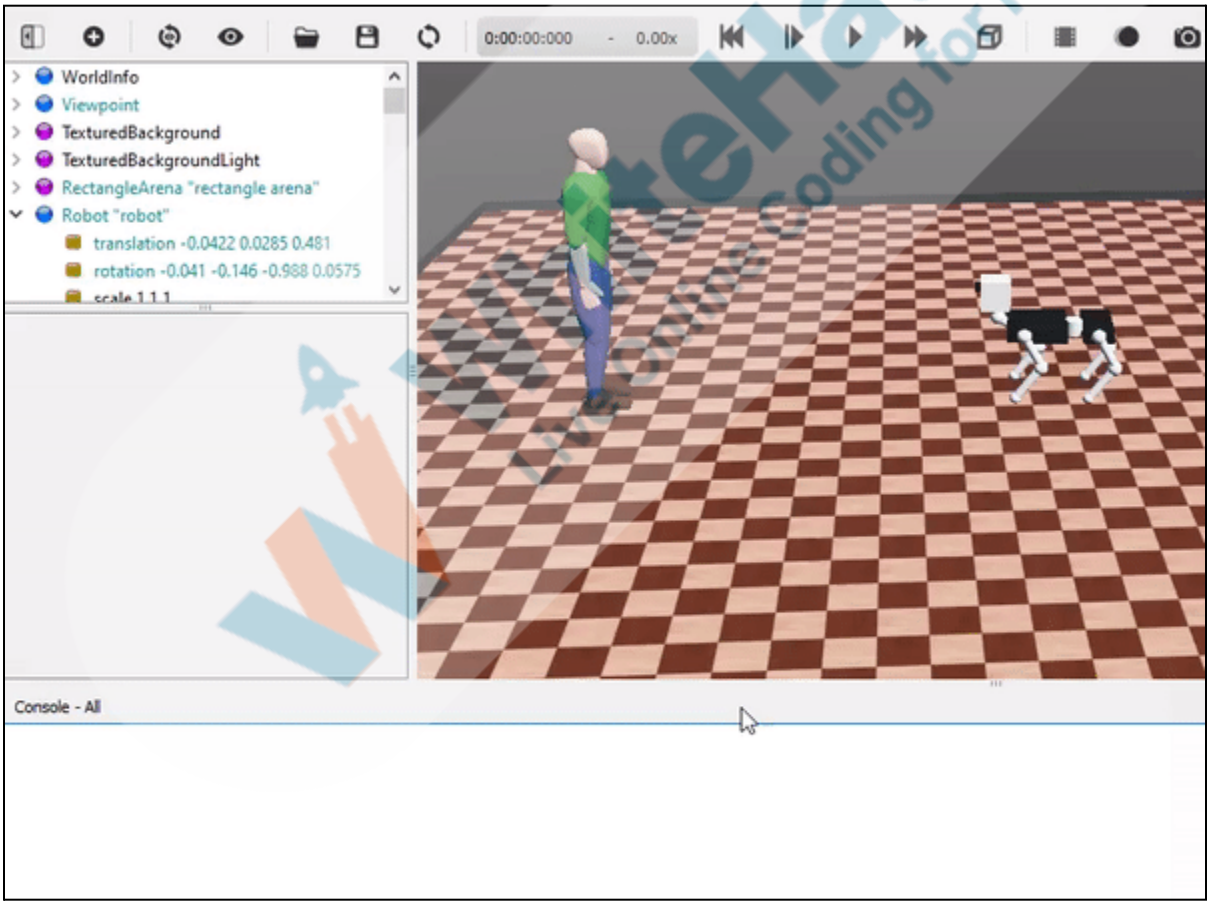
Reference code:

```
from controller import Robot

robot=Robot()
timestep=64

leg1=robot.getDevice("front_left")
leg2=robot.getDevice("front_right")
leg3=robot.getDevice("back_left")
leg4=robot.getDevice("back_right")

receiver= robot.getDevice("receiver")
receiver.enable(timestep)

flag=0

def add_delay(num_timestep):
   time_counter = 0
   while robot.step(timestep)  !=  -1:
      if time_counter >= num_timestep:
```

```
            break
        time_counter += 1

def walk(flag):
    if(flag%10==0):
        leg1.setPosition(-0.3)
    elif(flag%10==2):
        leg2.setPosition(-0.3)
    elif(flag%10==4):
        leg3.setPosition(-0.3)
    elif(flag%10==6):
        leg4.setPosition(-0.3)
    elif(flag%10==7):
        leg1.setPosition(0.2)
        leg2.setPosition(0.2)
        leg3.setPosition(0.2)
        leg4.setPosition(0.2)

def sit():
    leg1.setPosition(0)
    leg2.setPosition(0)
    leg3.setPosition(-0.37)
    leg4.setPosition(-0.37)
    add_delay(3)

def stand():
    leg1.setPosition(0)
    leg2.setPosition(0)
    leg3.setPosition(0)
    leg4.setPosition(0)
    add_delay(3)

while (robot.step(timestep) !=-1):
    if receiver.getQueueLength() > 0:
        received_data = receiver.getData().decode()
        print(received_data, type(received_data))
        receiver.nextPacket()

        if received_data == "sit":
            sit()
            received_data=" "
```

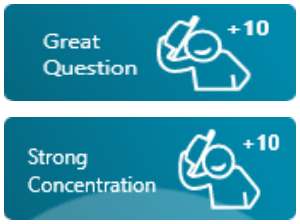
```
elif received_data == "stand":
    stand()
    received_data=" "

elif received_data == "walk":
    flag = flag + 1
    while(flag%30 != 0):
        walk(flag)
        flag=flag+1
        add_delay(3)
    received_data=" "
```

Reference Output:



Click here to view the output video.

Great work!	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	
Activity details **Following are the WRAP-UP session deliverables:** • Appreciate the student. • Revise the current class activities. • Discuss the quizzes.	
WRAP-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Explain the facts and trivia • Next class challenge • Project for the day • Additional Activity (Optional)	
FEEDBACK • **Appreciate and compliment the student for trying to learn a difficult concept.** • **Get to know how they are feeling after the session.** • **Review and check their understanding.**	
Teacher Action	**Student Action**
You get "hats-off" for your excellent work! In the next class, we will learn about motion files and a Humanoid robot.	*Make sure you have given at least 2 hats-off during the class for:* 

	
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks 	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Teacher Boilerplate Code	https://github.com/procodingclass/PRO-C294-Teacher-Boilerplate
Teacher Activity 3	Reference Code	https://github.com/procodingclass/PRO-C294-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/664b5762-4e9f-4aae-a846-85f135d2fe59.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C294-Project-Solution
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/502d62fb-c46d-414a-91d1-fa9291b42e1b.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C294-Student-Boilerplate