

CSCI-1100

Topics in Computing
University of Georgia

Lab Draft

NAO Robot: An Introduction to Python

Purpose: The purpose of this lab is to introduce you to programming and robotics!

In this lab, we are going to learn the following concepts:

- Programs and Algorithms
- Variables and Data Structures
- Expressions
- Conditional Statements
- Robotics

Part 0: Lab Introduction

Programs are what make computers useful. **A program is a finite sequence of instructions written in a particular language (called a programming language) that achieves some task.** We've seen many examples of programs like Windows, Mac OSX, email clients, web browsers, video games, text editors, and many others. Programs usually achieve one or multiple tasks. For example, an email client program will log you into an email server, send and receive email, and offer many other services like organizing your emails.

Before we can dive into programming, we have to take a step back and define an important concept in computer science called an **algorithm**. **An algorithm is a finite sequence of steps that achieves some task.** An algorithm is independent of any programming language, and it can be thought of as "a way to solve a problem" or "a set of directions to perform some task". The difference between an algorithm and a program can be subtle sometimes, but remember this: an algorithm is just a general solution to a problem or way to achieve a task whereas a program is a solution written in certain programming language that solves a problem or achieves some task.

In this lab, we are going to program a NAO robot using the programming language called Python! You have been hired at Athena Technologies® - an industry leading robotics corporation. Athena Technologies® specializes in programming NAO Robots. Athena Technologies has been contracted by the University of Georgia to determine the score of the next football game against the Florida Gators! In this lab, we will first come up with a list of steps that our robot will take, and then we will instruct our robot to do them. Thus, by the end

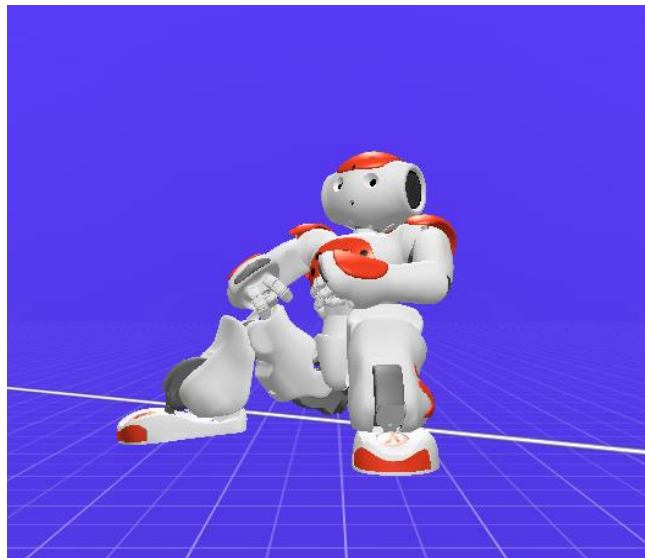
of the lab you will become a **programmer**, a person who translates an **algorithm** into a **program** that executes on a computer. Sounds easy, right? Let's get started!

NAO Robot

<https://www.aldebaran.com/en/humanoid-robot/nao-robot>
<https://www.aldebaran.com/en/humanoid-robot/nao-robot-working>

Python

<https://www.python.org/>
<https://www.python.org/about/apps/>



Part 1: The algorithm for ScoreBot

Before we jump right into programming our robot, we need to think about how we would solve this problem. Thus, we need to come up with an **algorithm** to follow. We can't program a robot to solve this problem unless we can think of how to do it ourselves!

Here are the general steps for an algorithm that we will use for this lab:

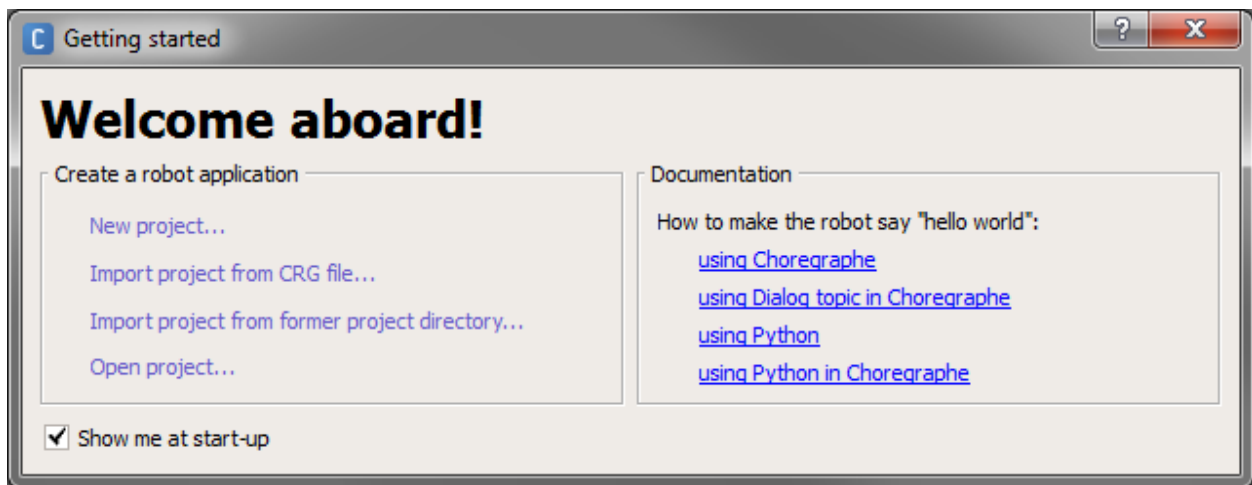
1. First, we will make NAO (our robot) stand up!
2. Secondly, we will need to come up with some scores of the game
 - a. We need to randomly generate a score for the Bulldogs

- b. We need to randomly generate a score for the Gators
3. Next, we need to compare the scores.
4. If the Bulldogs have a greater score than the Gators, then the robot should put his arms up and say “The Bulldogs Win!”
5. If the Bulldogs don’t have a greater score than the Gators, then the robot should sit down and say “The Bulldogs Didn’t Win.”

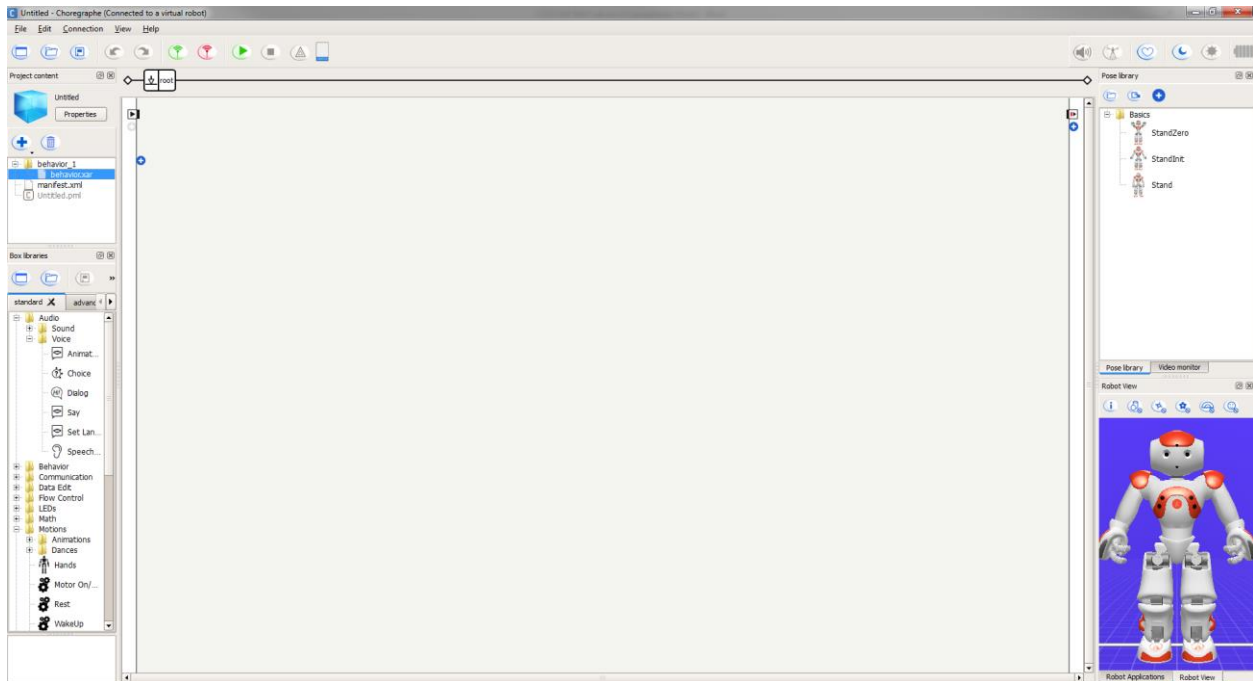
Part 2: NAO Robot’s IDE: Choregraphe

To start, we need to open up the software we will use to program our robot. This name of this software is Choregraphe. On a Windows machine, either click the windows key on the keyboard or click start on the desktop. Search for “Choregraphe” and open the program.

Upon opening Choregraphe, we are greeted with the following pop-up box:

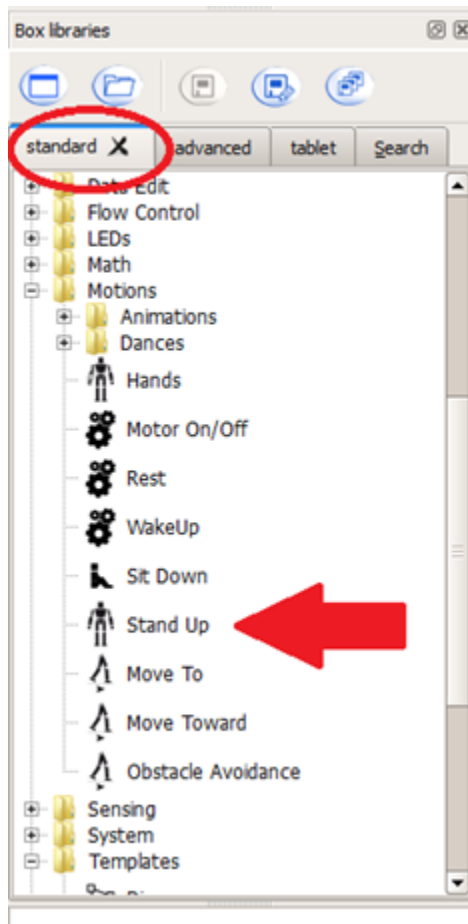


We are going to want to click “New Project”. Below is a screenshot of what you should see as a blank project. Choregraphe is both a drag-and-drop environment as well as a text editor where you can manually write code. For this lab, we will be doing a little of both!



Now, on the bottom right hand side of the screen is the view of your robot that we'll be programming! Across the very top of the screen, you can see all of your options for managing projects, undoing/redoin changes, connecting to a physical robot to test your code on it in real life, and lastly you have the start/stop commands for running your project. Working down the left hand side of the screen, we see the overview of our project, and before that we have some sample pieces of code we can use as a starting point for our project. We can call this section that contains these pieces of already done code the "Box Libraries" section. We'll use this heavily throughout the lab.

Let's begin by having our robot stand up. To do this, scroll down in the Box Libraries section of Choregraphe. Make sure you are on the standard tab, and keep scrolling until you see a folder called "Motions". Underneath that folder, we can see the "Stand Up" command. Below is a screenshot of what that looks like.



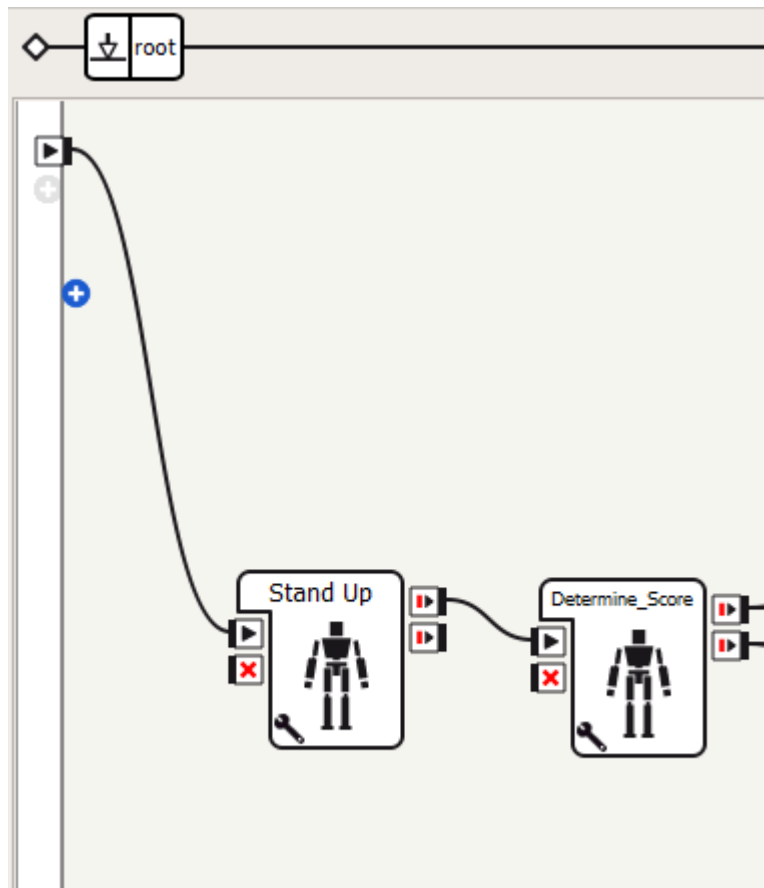
To use “Stand Up” in our project, click and drag it from the box libraries section into the main project area. Try and keep it somewhere in the top left of the project area. This will keep everything nice and neat. Now, the very next thing we want to do is grab the little black arrow attached to the top right side of the project area, and we want to drag that onto the matching black arrow of our “Stand Up” box. Below is a screenshot of what it looks like when you’ve completed this step.



Now that we know we can add actions for our robot to perform by dragging and dropping them from the Box Libraries section on the left hand side of the screen, we are going to add one last action for him to perform and we are going to edit and customize it to perform the task we need it to do!

Start by clicking and dragging another “Stand Up” from the Box Libraries section. Place it somewhere to the right of the first “Stand Up”. The second one is called “Stand Up (1)” to differentiate it from the first one that we already made. Let’s rename it! Right click on “Stand Up (1)” and click on “Edit Box”. Our robot is already standing up, so we just need this box to generate the scores and determine a winner. So, we are going to fully customize this action and make it our own. First, you should change the name of “Stand Up (1)” to “Determine_Score”. Giving our actions useful names is really important when you work at a company like Athena Technologies®. Next, edit the description of our box. Instead of the description talking about standing up, we want the description to say something along the lines of how it will determine the victor of the Georgia/Florida game. Leaving useful information like this as you work is a form of “Commenting”. Commenting is when programmers leave notes about their work so that when someone else looks at the project, they can see how things work and what they do. After editing the name and the description of our box, click “OK”. The very last thing we want to do right now is connect our “Stand Up” box to our “Determine_Score” box. You do this by clicking and dragging the success output from “Stand Up” and connecting it to the input of

“Determine_Score”. The success output is the topmost output on the right side of “Stand_Up”. Below is a screenshot of an output that you’ll want to connect into “Determine_Score”.



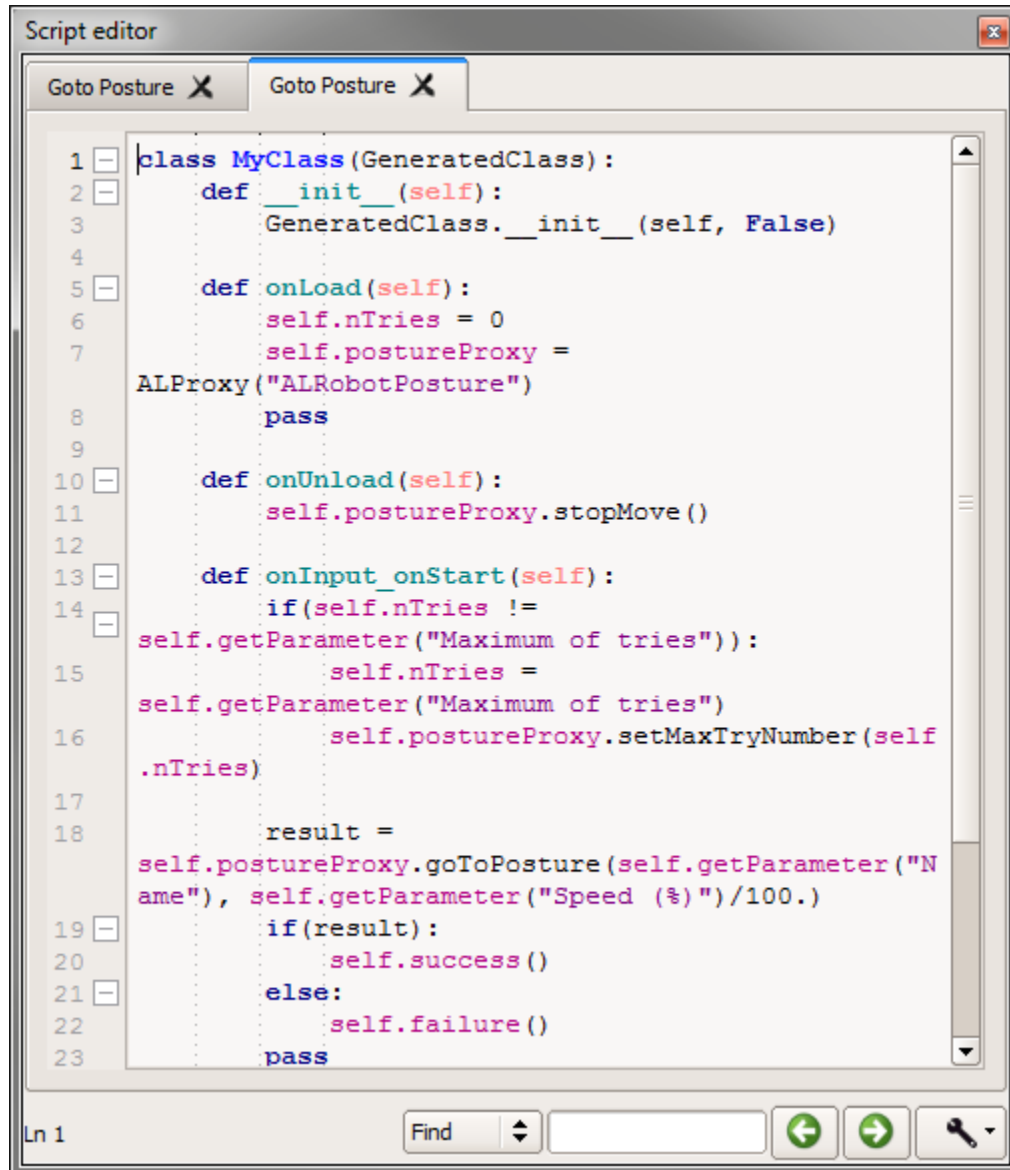
Once they are connected, we are all done with the drag and drop portion of the lab and can start programming in Python!

Part 3: Programming in Python

Now that we are done setting everything up in Choregraphe, it’s time to get started writing code in Python. As stated before, Python is a programming language and we will use it to compute the winner of the big game!

Start by double clicking on the Determine_Score box. This brings you down a level onto Determine_Score. In here, you can see everything that Determine_Score does. For now, it is filled with another box called “Goto Posture”. This is leftover from the “Stand Up” command that our box used to be. You can see how input comes in from the left, goes into the “Goto

Posture” box, leaves the box from an output, and then goes to the output on the right side of the screen area. You can see that the icon of this box is actually the Python logo! This box is actually a Python file, which we will edit. Double click on this box, and you will bring up some Python code! Here is the popup you will see:



The screenshot shows a 'Script editor' window with two tabs, both labeled 'Goto Posture' with a close button. The code is written in Python and defines a class 'MyClass' that inherits from 'GeneratedClass'. The code includes an initialization method, an 'onLoad' method that sets up a posture proxy, an 'onUnload' method that stops a move, and an 'onInput_onStart' method that handles movement attempts with retries. The code is as follows:

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self, False)
4
5     def onLoad(self):
6         self.nTries = 0
7         self.postureProxy =
8         ALProxy("ALRobotPosture")
9         pass
10
11    def onUnload(self):
12        self.postureProxy.stopMove()
13
14    def onInput_onStart(self):
15        if(self.nTries !=
16        self.getParameter("Maximum of tries")):
17            self.nTries =
18            self.getParameter("Maximum of tries")
19            self.postureProxy.setMaxTryNumber(self
20            .nTries)
21
22            result =
23            self.postureProxy.goToPosture(self.getParameter("Name"), self.getParameter("Speed (%)")/100.)
24            if(result):
25                self.success()
26            else:
27                self.failure()
28            pass
```

At the bottom of the window, there is a status bar showing 'Ln 1', a 'Find' field, and navigation buttons (back, forward, search).

This is the Python code for making our robot sit down. If you see on line 18, it makes a **variable** called “result”. In the following lines, it assigns a value to “result” with whether or not standing up was successful. This is one of the things that we will need to change.

1. Start by adding an import statement at the top of the Python file. The text of this import statement is "import random". This line let's our Python file know that it will use functions belonging to the random class. This is what will be used to generate random numbers for the score. Here is what it looks like:

```
1 import random
2
3 class MyClass(GeneratedClass):
4     def __init__(self):
5         GeneratedClass.__init__(self, False)
6
```

2. Next, we want to use our newly added random class to generate random numbers. We need to generate both the Bulldog's score as well as the Gator's score at random. First, we need to declare a **variable** to store the value of each score. Start with the Bulldog's score. Let's call this variable "resultDawgs". We will generate a random number of touchdowns for the Bulldogs to score, and then multiply that by 7 (the number of points in a touchdown). We will do the exact same for the Gators, except we will store their score in a different variable called "resultGators". Here is what it should look like:

```
19
20
21 #Random Number of Touchdowns for each team
22 resultDawgs = 7 * random.randint(0,10)
23 resultGators = 7 * random.randint(0,10)
24
```

On the right side of the expression, we have 7 (the number of points in a touchdown) multiplied by (*) a random integer between 0 and 10 (random.randint(0,10)).

3. Next, we want to compare the scores of the game! To do this, we will assign a Boolean value to a **variable**. Those are some strange words, so let's take it step by step. A Boolean can either be true or false. So, we are going to make a variable called "result" and it will either be true or false. If the value of result is TRUE, then it means the Bulldogs win! If the result is FALSE, then it means that the Bulldogs have lost. So, now that we know what we want this to do, let's translate that into code!

We will be using what's called an "if statement". If statements are special lines of code that have a boolean condition. If that condition is met (TRUE), then the lines underneath the if statement are executed. If the condition is FALSE, then the lines are not executed. What can sometimes follow an if statement is what's called an "else statement". An else

statement is similar to an if statement, however it does not require a condition because it accounts for everything that happens when the if statement is not met. Here is what it looks like in Python! You should copy this code into your lab:

```
25 result = resultDawgs > resultGators
26
27 ☐ if(result):
28     self.success()
29 ☐ else:
30     self.failure()
```

In English, line 25 makes our variable called “result”. If resultDawgs is greater than resultGators, then result is equal to TRUE. If resultDawgs is not greater than resultGators (either a tie or a loss), then result is set to FALSE. In line 27, we use an if statement. If the condition is TRUE, then the following line is executed. When the condition is false, it jumps to the else statement and executes that line of code instead. Lines 28 and 30 are what our Python code returns. In English: if the Bulldogs win, our program will end with success. If the Bulldogs do not win, our program will end with failure.

- 4.** We are all done with our Python code! Congratulations, you are now a Python programmer! The following screenshot will show you what your final Python code will look like:

Script editor

Goto Posture X Goto Posture X

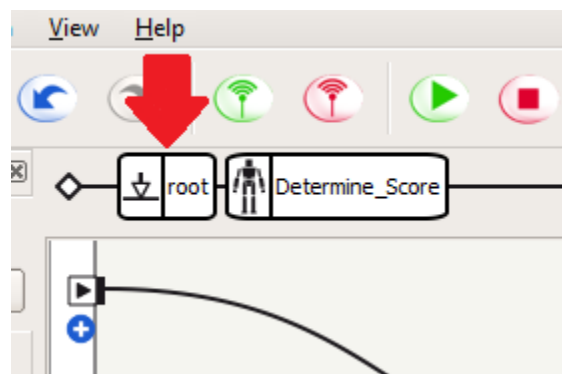
```
1 import random
2
3 class MyClass(GeneratedClass):
4     def __init__(self):
5         GeneratedClass.__init__(self, False)
6
7     def onLoad(self):
8         self.nTries = 0
9         self.postureProxy =
ALProxy("ALRobotPosture")
10         pass
11
12     def onUnload(self):
13         self.postureProxy.stopMove()
14
15     def onInput_onStart(self):
16         if(self.nTries !=
self.getParameter("Maximum of tries")):
17             self.nTries =
self.getParameter("Maximum of tries")
18             self.postureProxy.setMaxTryNumber(self
.nTries)
19
20         #Random Number of Touchdowns for each team
21         resultDawgs = 7 * random.randint(0,10)
22         resultGators = 7 * random.randint(0,10)
23
24         result = resultDawgs > resultGators
25
26         if(result):
27             self.success()
28         else:
29             self.failure()
30         pass
31
32     def onInput_onStop(self):
33         self.onUnload() #~ it is recommended to
call onUnload of this box in a onStop method, as
the code written in onUnload is used to stop the
box as well
34
35         pass
```

Ln 35 Find

Part 4: Programming Reactions

We are all done with our Python code! Now that our Robot will determine the winner of the game, we need to program our robot to react to the result! So, if the Bulldogs win, we want our robot to put his hands up and say “Dawgs win!” If the Bulldogs do not win, we want our robot to sit down and say “Boo Hoo, the Dawgs lose.” Let’s program our robot to execute these reactions.

This is the screen you should now be on. After closing our Python code’s popup box, we are still inside the Determine_Score box. Let us return to the root of our project. Do this by clicking on “root” at the top of the project area. Here is what that button looks like:

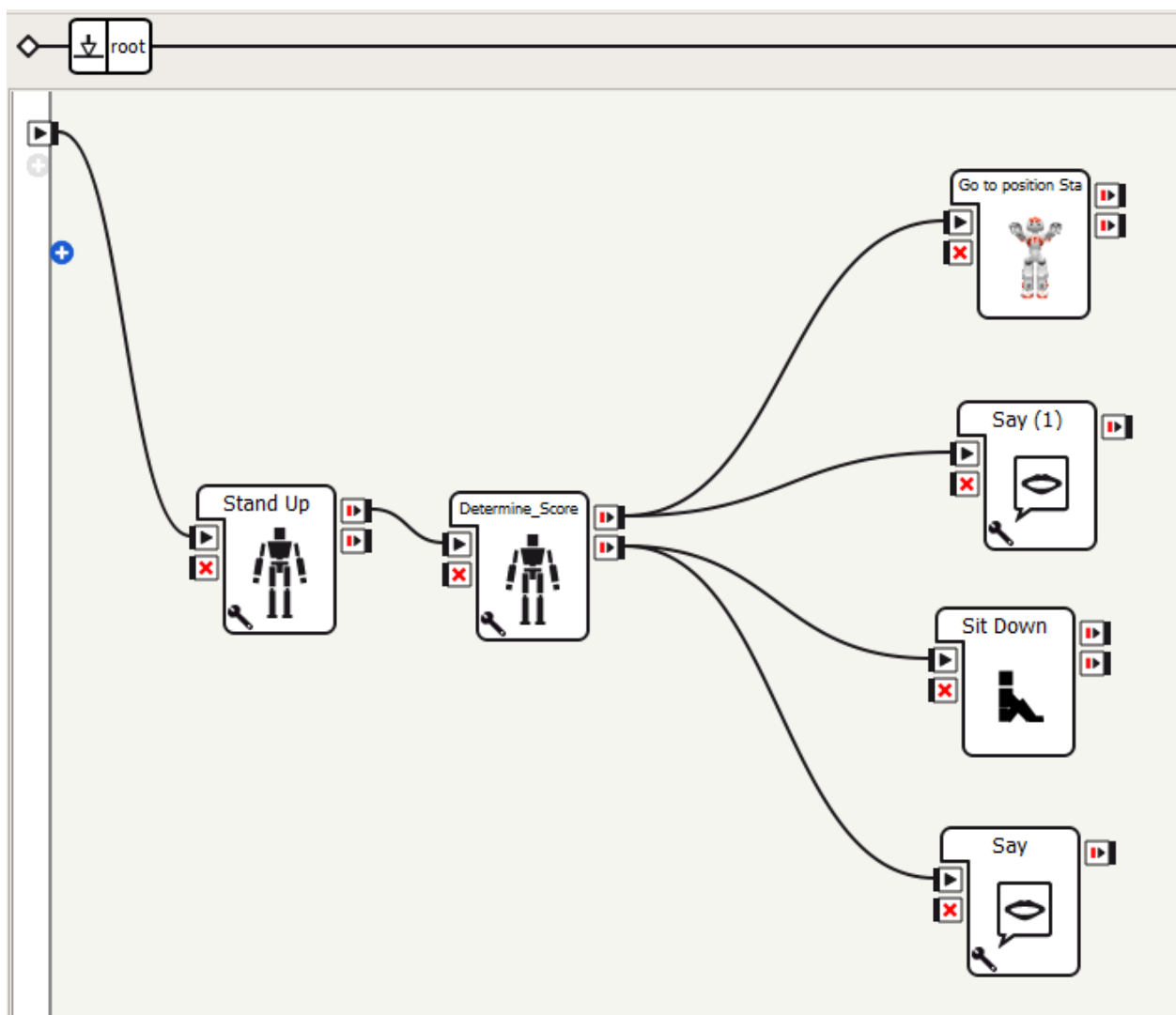


After clicking on “root”, we are returned to the main level of our project! At this point, we only have to program our result! As stated earlier in the lab, the outputs for each box are the buttons on the right side of the box. The one on top corresponds to a success output, the one on the bottom corresponds to a failure output. Remember above how when the Bulldogs win, the Determine_Score is successful? And if they do not win, the box returns with a failure? We will use this to program the robot’s reactions.

Here is what we will need to do:

1. Similarly to how we found the “Stand Up” box, we need to find the “Sit Down” action from our list. Click and drag it somewhere to the right of our Determine_Score box.
2. We also need another box. This one is called “StandZero”. StandZero can be found in the Pose library at the top right side of Choreographe. This box will raise our robots arm.
3. Lastly, we need 2 “Say” boxes. These can be found in the same section as “Stand Up” and “Sit Down”, but under the folders Audio->Voice.

Thus, with these 4 new boxes, we will need to connect them like so:



So, the SUCCESS output from Determine_Score will send input to our “StandZero” box as well as one of the “Say” boxes. The FAILURE output from Determine_Score will send input to our “Sit Down” box as well as the other of the “Say” boxes.

Part 5: Finishing Up

The very last thing to do once everything is connected is to tell our robot what to say! Remember, whatever box is connected to the success output (the topmost output of Determine_Score) will need to say “Dawgs Win!” The “Say” box connected to the failure output (the one on bottom) will need to say “Boo Hoo, the Dawgs lose.” To change what the robot says, simply double click on a “Say” box, and you can change the text of the box from there. Do this for both of our “Say” boxes, and you have finally finished your first project at Athena Technologies®!

Part 6: Conclusion

So, let’s reflect on what we accomplished during this lab. We learned key programming concepts as well as the syntax of a very popular programming language called “Python”. You have successfully come up with an algorithm, and had a robot to perform your instructions. Thus, you can officially call yourself a programmer! The NAO robot is a very powerful tool, and there is nothing stopping you from experimenting with the NAO robot and creating your own programs! Upon completing this lab, you know everything you need to get started!