Lesson 1: Loco-motion

Aim

Welcome to your first lesson in Misty Python! The aim of this lesson is to introduce you to some of the basics of building interactions with Misty using Python, starting with movement and locomotion. If you have any questions about the individual APIs to trigger locomotion and movement, you can check out the Motion and Mobility section in Python Elements.

Motion Sequences

Challenge 1: Hello again human!

If you've completed the Blockly Lessons you might remember that Misty started her first lesson by teaching you how to make her greet humans. In this challenge you can rebuild the Movement blocks using Python and modify Misty's head and arm positions. Let's dive into Misty Python!

Below are three lines of API calls that you can use, the API `misty.move_head` on the first line controls Misty's head movement. Like the Head movement block in Blockly, the parameters are pitch, roll and yaw. If you don't recall the range of values for this API you can check them out in Motion and Mobility.

The second line is an arm movement API called `misty.move_arm`, in the parameters you need to specify which arm Misty has to move and in to which position, the first value is a string and can be "right", "left" or "both" and the second is the value to define the degree of movement.

The third line is another arm movement line with the API `misty.move_arms`, which allows you to control two arms simultaneously and set them to different degrees. Let's try using these APIs to have Misty wave a hello again!

```
from mistyPy.Robot import Robot

misty = Robot()

#first line
misty.move_head(pitch= 0, roll= 20, yaw= 0)

#second line
misty.move_arm(arm= "both", position= 0)

#third line
misty.move_arms(leftArmPosition= 0, rightArmPosition= 70)
```

Challenge 2 : Remember to sequence

Just like in Blockly, when you build a sequence of two or more movements Misty needs time to achieve each position and execute the code in a consecutive order. If you don't give her time between movements, she will try to execute the movements at the same time. While in Blockly we could simply insert a timer block, in Python you will need to import a library with the time functions you need.

A library is a group of functions created be programmers for other programmers to use. It's a really powerful tool because you can implement predefined functions instead of building your own from scratch. In this case, we'll use the library "time", where we can find many functions related to timing sequences.

To use a the time library you need to import it as shown in the example below and then call the specific function you want to use.

```
import time

time.sleep(#timeinseconds)
```

In this case we will use the `time.sleep` function which works much like the timer block in Misty Blockly. The parameter's unit is in seconds.

Now try out timing your sequence to have Misty wave her right arm up and down.

```
from mistyPy.Robot import Robot
import time   # importing the library

misty = Robot()

misty.move_head(0, 30, 0)
misty.move_arms(70, -50)
time.sleep(0.4)  #using the function in the library
misty.move_arms(70, 0)
time.sleep(0.4)  #sleep for 0.4 seconds or 400 milliseconds
misty.move_arms(70, -50)
time.sleep(0.4)
misty.move_arms(70, 0)
```

If you want you can also define which arm you want to use.

```
from mistyPy.Robot import Robot
import time

misty = Robot()

misty.move_head(0, 30, 0)
misty.move_arm("left", 70) # choosing Misty's arm and its position
misty.move_arm("right", -50)
time.sleep(0.4)
misty.move_arm("right", 0)
time.sleep(0.4)
misty.move_arm("right", -50)
time.sleep(0.4)
misty.move_arm("right", 0)
```

Challenge 3: Loop your sequences

Just like in Blockly you can also create loops in Python to save space and compact your code. Let's give it a try by having Misty repeat your sequence 5 times!

There are man ways you can code loops in Python, for this example you can use the `for i in range ()` method. Within the brackets you can choose the number of times you want the sequence to be repeated.

To create a loop with this method you need 4 elements:

- initialization: where you define the variables that you'll use in the loop. In this case the "i"
- check: where you control if the loop is verified and so go ahead. With this method, you don't need to insert the check because it's automatic.
- body: where you have all the action that your loop does. In this code all that's after the ":" and one tab away from the "for"
- update: where you update your variables, in this way your variable will be increased and there will be a moment where the check is not verified anymore, in this way, you can stop the loop. With this method, the update is automatic.

To exit from the loop and insert commands after the loop you can keep writing your code on the same line as the "for"

```
from mistyPy.Robot import Robot
import time
misty = Robot()

for i in range(5): #init, check, update
    misty.move_arms(70, -50) #body
    time.sleep(0.4)
    misty.move_arms(70, 0)
    time.sleep(0.4)
#other actions outside the loop
misty.move_arms(0, 0)
```

Driving

Challenge 4: Need for speed

Now that you've mastered Misty's arm and head movement APIs, let's explore how to make her drive using the `misty.drive_time` and `misty.drive_arc` API calls.

In the code example below, Misty is driving forward for 3 seconds, then she does a turn and goes back to her initial position. In the first two lines of code you can see that the parameters clearly defined, while in the second set of lines they are shown as regular values. In the future you can choose to highlight the parameters or memorize their order and write values. Now let's test to see how she executes the pattern on the ground, if you want her to perform it like a street car, you can always satisfy your need for speed by increasing the velocity.

```
from mistyPy.Robot import Robot

misty = Robot()

misty.drive_time(linearVelocity= 80, angularVelocity= 0, timeMs= 3000)
misty.drive_arc(heading= 180, radius= 0.5,timeMs= 4000)
misty.drive_time(80, 0, 3000)
misty.drive_arc(180, 0.5, 4000)
```

Challenge 5: Plan a path

Misty's has many more driving options than just `drive_time` and `drive_arc`. This range of driving function allow to create path plans for Misty. In later lessons you will also learn how to use AR tags and QR code to help Misty navigate around your space.

Let's start by creating a simple path plan that uses `misty.drive_track` and `misty.drive_heading`. In this case Misty will go backwards for two seconds then will turn left, she'll move just her left track for one second, forward for one meter, and then drive like there was a curve for five seconds more.

When you want Misty to stop driving or moving between different stages of your path plan, you will need to use the `misty.stop` function which will stop all of her motors.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.drive(linearVelocity= -70, angularVelocity= 0)
time.sleep(2)
misty.stop()
misty.drive_arc(90, 0, 3000)
misty.drive_track(leftTrackSpeed= 100,rightTrackSpeed= 0)
time.sleep(1)
misty.stop()
misty.drive_heading(heading= 0,distance= 1,timeMs= 4000)
misty.drive_time(20, 60, 5000)
```

Challenge 6: Returning to the red carpet

Remember when Misty went out on a red carpet in the Misty Blockly Lessons? Let's try building that sequence again using all the Misty API's you've learned so far. You can use the following example for inspiration.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.drive(70, 0)
misty.move_head(0, 0, 20)
misty.move_arms(70, -50)
time.sleep(0.4)
misty.move_arms(70, 0)
time.sleep(0.4)
misty.move_arms(70, -50)
time.sleep(0.4)
misty.move_arms(70, 0)
misty.move_head(0, 0, 0)
time.sleep(0.5)
misty.stop()
```

Up Next 🐥

PAGE
Lesson 2 : Build a character

Last updated 2 months ago

Lesson 2 : Build a character

Aim

The aim of this lesson is to introduce you to the concept of building robot characters using Misty's API calls that allow you to enable Misty's speech, as well as display and vocalize expressions. If you have any questions about the APIs in this lesson you can check out the Display and LED and Speech and NLP sections.

Expressions and Speech

Challenge 1: Eyes to the skies

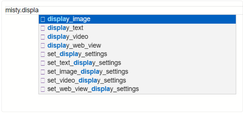
Imagine that Misty is a robot astronaut visiting Mars and she raised her head to look at the stars, suddenly a UFO flies by. Is it a plane, a helicopter, an alien bird? The first step is to decide how you want Misty to express her reaction, by using the `misty.display_image` API you can choose which default expressions Misty should show. In this example we chose 'Terror'. You can find a full list of them here: Image files. Now try out combining the display and movement APIs to build the described sequence.

For more information about the display and its parameters, you can check the Display and LEDsection in Misty Python API. The parameter "alpha" represents the opacity of the image.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

#the alpha parameter determines the opacity of the image
misty.display_image(fileName= "e_Contempt.jpg", alpha= 1)
time.sleep(1)
misty.move_head(-40, 0, 0)
misty.move_arm("both", -90)
time.sleep(2)
misty.display_image("e_Terror2.jpg")
```

Misty' has many other display capabilities which you can uncover by typing the keyword "display". If you want her to display a video or text instead of an image, you can use the same arguments within the parameters.



Challenge 2: I spy with my little eye

Great job! You have built your first robot character sequence, but it's still missing an important human element to really bring your character to life. The `misty.speak` API works much in the same way as the Speak block in Blockly and shares many of the same parameters. You can choose what Misty will say, in what language and with which pitch. Misty uses US English by default if nothing else is specified. Let's try inserting the API into the sequence and have Misty tell you what she sees.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.display_image("e_Contempt.jpg")
time.sleep(1)
misty.move_head(-40, 0, 0)
misty.move_arm("both", -90)
misty.speak(text= "I spy with my little eye", pitch= 0.8)
time.sleep(0.5)
misty.display_image("e_Terror2.jpg")
misty.speak("a UFO!!")
```

Challenge 3: Uh oh!

You can also have Misty emphasize her reaction with one of her vocal expressions by using the `misty.play_audio` API. You can find all Misty's default audio in Audio files and if you've uploaded audio files on Misty in Blockly, you can use their names in the parameters. How would you customize it?

As shown in the example below, the first parameter is a string, and it contains the name of the file you want to use, the second is a float and it represents the volume.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.display_image("e_Contempt.jpg")
time.sleep(1)
misty.move_head(-40, 0, 0)
misty.move_arm("both", -90)
misty.display_image("e_Terror2.jpg")
misty.speak("I spy with my little eye")
time.sleep(0.5)
misty.speak("a UFO!!")
time.sleep(0.5)
misty.play_audio(fileName= "s_PhraseUhOh.wav",volume= 80)
```

Language

Challenge 4: Can you speak my language?

What if your robot astronaut ends lands by accident in a different country and suddenly you need Misty to speak a different language? In the Speak Block the language could be selected from a drop-down list, however in Python you will need to set the language as voice parameter with a string of code. You can find the strings for each language in Languages. Let's try having Misty repeat the previous sequence in Italian or in one of your favorite languages.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.display_image("e_Contempt.jpg")
time.sleep(1)
misty.move_head(-40, 0, 0)
misty.move_arm("both", -90)
misty.display_image("e_Terror2.jpg")
misty.speak("Spio con i miei occhietti", voice= "it-it-x-itb-local")
time.sleep(0.5)
misty.speak("un UFO!!", voice= "it-it-x-itb-local")
time.sleep(0.5)
misty.play_audio(fileName= "s_PhraseUhOh.wav",volume= 80)
```

Chest LED

Challenge 5: Houston we have a problem

To emphasize your robot character's expressions even more you can use the `misty.change_led` API to control the Chest LED. For example you can add a yellow alert light to the previous example to emphasize Misty's terror of seeing the UFO. You can also use the `misty.transition_led` API to cycle between different colors.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.display_image("e_Contempt.jpg")
time.sleep(1)
misty.move_head(-40, 0, 0)
misty.move_arm("both", -90)
misty.speak(text= "I spy with my little eye", pitch= 0.8)
time.sleep(2)
misty.display_image("e_Terror2.jpg")
misty.change_led(red= 255, green= 255, blue=0)
misty.speak("a UFO!!")
misty.play_audio(fileName= "s_PhraseUhOh.wav",volume= 80)
misty.transition_led(255,0,0,255,255,0,"blink")
misty.speak(text= "Houston, we have a problem", pitch= 0.8)
```

Challenge 6: E.T go home

Remember how Misty created a disco ball effect in Blockly Lessons? We can adapt the same sequence to create a scenario where your robot astronaut gets sucked up by a colorful beam of light from the UFO.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.move_head(40,0,0,100)
time.sleep(0.5)
misty.move_head(-40,0,0,100)
time.sleep(0.5)
misty.move_head(0,0,0,100)
misty.move_arm("both",-60,100)
misty.display_image("e_EcstacyHilarious.jpg")
misty.play_audio("s_PhraseNoNoNo.wav",60)
misty.change_led(red= 0, green= 255, blue=40)
time.sleep(0.2)
misty.change_led(red= 150, green= 30, blue=0)
time.sleep(0.2)
misty.change_led(red= 48, green= 42, blue=79)
time.sleep(0.2)
misty.change_led(red= 231, green= 25, blue=41)
time.sleep(0.2)
misty.change_led(red= 94, green= 200, blue=194)
time.sleep(0.2)
misty.change_led(red= 3, green= 147, blue=240)
time.sleep(0.2)
misty.change_led(red= 149, green= 36, blue=236)
time.sleep(0.2)
misty.stop()
```

Challenge 7: Compact your code

Inserting values for each of the color parameters and adding new lines of code can be tiresome, this is where importing libraries can be incredibly helpful. By importing the "random" library you can extract a random number in a range from 0 to 255. in this way yo can randomize the color values, save lines of code and space for the sequence.

With a cleaner code and the ability to randomize your values, you can now generate how many colors you want without creating an enormous code. To do this, you will need to create a loop and insert the `random.randrange` function inside your `misty.change_led` API as shown in the example below. We can select the range and Misty will do the rest. Amazing isn't it?

```
from mistyPy.Robot import Robot
import random
import time
misty = Robot()

misty.move_head(40,0,0,100)
time.sleep(0.5)
misty.move_head(-40,0,0,100)
time.sleep(0.5)
misty.move_head(0,0,0,100)
misty.move_arm("both",-60,100)
misty.display_image("e_EcstacyHilarious.jpg")
misty.play_audio("s_PhraseNoNoNo.wav",60)
for i in range (100):
    misty.change_led(random.randrange(0, 255), random.randrange(0, 255),
random.randrange(0, 255))
    time.sleep(0.2)
misty.stop()
```

Up Next

PAGE
Lesson 3: Create memories

Last updated 2 months ago

🧠 Lesson 3: Create memories

Aim

The aim of this lesson is to learn how to manage content on Misty and access external media online. By the end of the lesson you will be able to capture content, upload custom images, video and audio as well as stream online content directly on your Misty's display. If you need detailed information about the API calls for this lesson you can check out Record Assetsand Display and LED.

Capturing content

Challenge 1: Snap a photo with Misty

Misty is a master at capturing information about her surroundings, especially with her visor camera. With the `misty.take_picture` API you can have Misty take a photo of you and objects in your room. Let's try to snap a picture with Misty and display it. In the parameters you can define the image name, choose to display it automatically, set its resolution and even overwrite it if you want snap a new version of the same image.

```
from mistyPy.Robot import Robot
misty = Robot()

misty.take_picture(base65=True, fileName="myportrait", width=3200,height=
2400,displayOnScreen=True,overwriteExisting=False)

#short version
# misty.take_picture(True,"myportrait",3200,2400,True,False)
```

Upload content

Challenge 2: A new pair of eyes

As you've probably discovered in Misty Studio, one of Misty's greatest talents is display customizability, allowing you to create your very own robot characters by uploading custom eyes and expressions. Typically local files on your computer or tablet can be uploaded in the Expressions tab under Explore in Misty Studio or by running the Upload Block in Blockly. But what if you want to give Misty your own customized eyes using Python code?

To upload an new pair of eyes directly from your computer in Python you would need to create a new programming environment outside of Misty Studio, since this is a more complicated process that we will explore in later lessons, the quickest way to upload images is using HTTP endpoints from your cloud such as Google Drive or directly from a website. To achieve this we will need to rely once more on the power of libraries. For Misty to read and encode data from an HTTP endpoint you will need to import the base64 library and use the `misty.save_image` and `misty.display_image` API calls. Let's try it out!

```
from mistyPy.Robot import Robot
import base64
import requests

#binary-to-text encoding schemes
misty = Robot()

# Encode the image as base64 and UTF-8 so it's safe for the API,
encoded = result =
base64.b64encode(requests.get("https://static.wixstatic.com/media/d181e9_b56a1c7cd810474
2af8b00cc716caacd~mv2.gif").content).decode("utf8")

# Send it to Misty with matching dimensions. In this case, displaying immediately and
overwriting any existing file
misty.save_image("customeyes.gif", encoded, 640, 480, False, True)
time.sleep(2)
misty.display_image("customeyes.gif")
```

Challenge 3: Create a robot movie theater

As you continue to design your robot character and interactions with Misty, don't forget to document all the fun moments. You can do this by letting her record them or by uploading your own footage of your interactions. You can also fill her memory banks with your favorite short movies and create a robot cinema. The API calls for uploading and displaying videos work in the same way as for images.

Note: Accepted video file types are .mp4 and .wmv. Maximum file size is 6 MB.

```
from mistyPy.Robot import Robot
import base64
import requests
import time

#binary-to-text encoding schemes
misty = Robot()

# Encode the video as base64 and UTF-8 so it's safe for the API,
encoded = result =
base64.b64encode(requests.get("https://video.wixstatic.com/video/d181e9_837968f95e3f4cd2
a6210ec36ff87f58/360p/mp4/file.mp4").content).decode("utf8")

# Send it to Misty with matching dimensions. In this case, displaying immediately and
overwriting any existing file
misty.save_video("mistywow2.mp4", encoded, False, True)
time.sleep(2)
misty.display_video("mistywow2.mp4")
time.sleep(2)
misty.stop(1)
```

Challenge 4: Add a new vocal expression

In some cases you maybe want to have a Misty play your music or use custom vocal expressions for you robot character. In that case you can download an audio file to Misty's storage from a cloud drive and try out playing it.

```
from mistyPy.Robot import Robot
import base64
import requests
import time

#binary-to-text encoding schemes
misty = Robot()

# Encode the image as base64 and UTF-8 so it's safe for the API,
encoded = result =
base64.b64encode(requests.get("https://assets.mixkit.co/active_storage/sfx/93/93-
preview.mp3").content).decode("utf8")

# Send it to Misty with matching dimensions. In this case, displaying immediately and
overwriting any existing file
misty.save_audio("newvoice1.mp3", encoded, False, True)
time.sleep(1)
misty.play_audio("newvoice1.mp3",30)
```

Stream content

Challenge 5: Stream you favorite content to Misty

Now that you've mastered how to upload new content to Misty, let's learn how to program Misty to stream online content directly on her display. To achieve this you can use the misty.display_web_view API, this is a great feature if you want Misty to show you the latest news, weather, statistics or other live information. What would you choose to stream?

Misty uses the default webview layer settings the first time she draws content with the `DisplayWebView` command.

```
from mistyPy.Robot import Robot
import time
misty = Robot()

misty.speak("Good morning my human friend, do you wanna know how cold it is outside
today? Take a look!")
time.sleep(2)
misty.display_web_view("https://www.yr.no/en/forecast/daily-table/2-
5574991/United%20States/Colorado/Boulder/Boulder")
time.sleep(5)
misty.stop(1)
```

Important! Displaying webviews can consume a lot of computational resources. If you notice Misty's performance decrease while multiple webviews layers are active, you may consider deleting one or more webview layers. You can use the `SetWebViewDisplaySettings` command to adjust the settings and change the appearance for a specific webview layer. Issuing a `SetWebViewDisplaySettings` command redraws the updated webview layer on Misty's display.

Up Next 👇

PAGE
Lesson 4: Event skills

Last updated 2 months ago

⚒️ Lesson 4: Event skills

Aim

The aim of this lesson is to learn how to make Misty more aware of her surroundings using event-based skills. By the end of the lesson, you will be able to build basic skills in Python and utilize most of Misty's sensory capabilities to create more complex social interactions with your robot character.

Defining Events

Challenge 1: Import, register and define an event

You've probably already discovered in Blockly all the fun interactions you can build using Misty's capacitive touch and bump sensors. Now let's try building them in Python!

To use events in Python you will need to import the relevant library that contains all the events that Misty can trigger. This library works as any other library in Python and it's called "mistyPy.Events", it contains the APIs for your events such as BumpSensors, Facerecognition, ARTagdetection etc. You can find out more about them in the Events section in Python Elements. Let's start with importing the library.

```
from mistyPy.Robot import Robot #libraries
from mistyPy.Events import Events
```

Once you've imported this library the next step is to register the event you want in your code using the `misty.register_event` API. To have Misty continously monitor her sensors you can use the equivalent to the Run Until Stopped block in Blockly called `misty.keep_alive.`

```
from mistyPy.Robot import Robot #libraries
from mistyPy.Events import Events

#events definitions
misty.register_event(event_name='bump_event', event_type=Events.BumpSensor,
callback_function=bumped, keep_alive=False)

#"run until stopped" of blockly
misty.keep_alive()
```

In the first parameter you need to insert the the name of the event you want to use. The second parameter identifies which event in that category you want to trigger. You need to use one of the names provided in Events. The third parameter is the callback function which contains the name of the function you need to use to define which commands Misty should execute when an event is triggered. The last parameter is `keep_alive`, if you set it to "False", once your event has been triggered the code will stop, instead, if you set it to "True" you can decide when to stop your code by clicking the Stop button. In the default line, the last parameter is set as False.

Finally, the third step is to define and create the sensor functions associated with your event using `def function_name(data)` and `print(data)`. This can be placed between your library import and the registered event. For your sequence to be contained within the function you need to place it after `print(data)`.

```
from mistyPy.Robot import Robot #libraries
from mistyPy.Events import Events

#function
def bumped(data):
    print(data)
    # Function code here
    # You can have multiple lines of code inside the function
    # The function's logic goes here

#events definitions
misty.register_event(event_name='bump_event', event_type=Events.BumpSensor,
callback_function=bumped, keep_alive=False)

#"run until stopped" of blockly
misty.keep_alive()
```

Event skills

Challenge 2: Python event skills with a human touch

Great work! Now that you've successfully created a template for your event, you can start building event based skills in Python for your robot character using the skill tree concept that we covered in Blockly Lessons. Human touch, like a handshake or pat on the back, is an important element to social interactions, and human-robot interactions are no exception.

Using the template you created, you can now register other events such as the capacitive touch sensor event in the example below. When you are building functions you can choose to specify which sensors you want to use from a group, if you don't specify the sensor Misty will trigger the function when any of the sensors in the group are triggered, similar to when you select *"any"* when creating sensor events in Blockly. Let's try using all of the the movement and expression API calls you've learned so far to have Misty respond to your touch.

```
from mistyPy.Robot import Robot #libraries
from mistyPy.Events import Events

misty = Robot()

def bumped(data):   #functions
    print(data)
    misty.display_image("e_Love.jpg", 1)
    misty.move_head(0, -10, 0, 50)
    misty.move_arms(90, -70, 50, 50)
    misty.change_led(255, 0, 0)

def touched(data):
    print(data)
    misty.display_image("e_Joy.jpg", 1)
    misty.move_head(0, 0, 0, 50)
    misty.move_arms(-40, 90, 50, 50)
    misty.change_led(0, 255, 0)

#events definitions
misty.register_event(event_name='bumped', event_type=Events.BumpSensor,
callback_function=bumped, keep_alive=True)
misty.register_event(event_name='touch', event_type=Events.TouchSensor,
callback_function=touched, keep_alive=True)

#"run until stopped" of blockly
misty.keep_alive()
```

Challenge 3: Expand your event skill tree

Congratulations! You have built your first event skill tree in Python. Now you can grow your skill tree by creating more branches in your interaction with Misty using individual sensors in a group. To do this you will need to build a condition in your function that will help Misty verify which sensor to use.

For example in the bump sensor function, to select a specific bumper in the group you need to create a condition using an if statement where you specify which of them you want to use:

```
if data["message"]["sensorId"] == 'bfr':
#'bfr' is short for Front-Right Bumper
```

For the bump sensors and the touch sensor, you can use their name ID. You can find all the name ID's for the sensors in Sensor Events

When you create a condition it's always good practice to account for the possibility that it may not be verified due to syntax errors in your code. For example we can use an else statement to have Misty show a red light when something goes wrong.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events

misty = Robot()

def bumped(data):
    print(data)
    if data["message"]["sensorId"] == 'bfr': #condition
        misty.drive_time(0, -50, 8000)
        misty.change_led(0, 255, 255)
        time.sleep(1)
        misty.drive_time(30, 0, 5000)
        misty.change_led(255, 255, 0)
    else: #what happens if the condition is not verified
        misty.change_led(255,0,0)

def touched(data):
    print(data)
    if data["message"]["sensorPosition"] == 'HeadFront' :
        misty.change_led(0, 255, 0)
        misty.drive_time(30, 0, 5000)
        misty.display_image("e_EcstacyHilarious.jpg", 1)
    else:
        misty.change_led(255,0,0)

misty.register_event(event_name='touch', event_type=Events.TouchSensor,
callback_function=touched, keep_alive=True)
misty.register_event(event_name='bumped', event_type=Events.BumpSensor,
callback_function=bumped, keep_alive=True)

misty.keep_alive()
```

Challenge 4: Compact your event skill tree

As you grow your event skill tree you will find that your event functions can take up a lot of space. To save space and time try compacting your skill tree using the `elif` (else if) statement. This will allow you to contain all of the conditions for each sensor within a single function group as shown in the example below.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events

misty = Robot()

misty.speak("What's your favourite colour?", 1)
misty.move_head(0, 0, 20, 50)
misty.display_image("e_Contempt.jpg", 1)

def bumped(data): # function with more condition
    print(data)
    if data["message"]["sensorId"] == 'bfr':
        misty.change_led(0,0,255)
    elif data["message"]["sensorId"] == 'bfl':
        misty.change_led(255,0,0)
    elif data["message"]["sensorId"] == 'brr':
        misty.change_led(255,0,255)
    elif data["message"]["sensorId"] == 'brl':
        misty.change_led(0,255,0)
    else:
        misty.change_led(255,255,255)

misty.register_event(event_name='bumped', event_type=Events.BumpSensor,
callback_function=bumped, keep_alive=True)

misty.keep_alive()
```

Event message

Challenge 5: Generalize your event skill tree

If you've worked with event messages in Blockly you may remember that it's an elegant way to compact and generalize your event skill. Instead of writing values in each condition within a function group, you can simply use an event message to return the value from the sensor that you have triggered. You can use the data from an event as strings in your code and make it more flexible and slim. Event messages can be used in a number of events such as Face Recognition, Object Recognition, BumpSensor, TouchSensors, etc. For example if we use the `misty.speak` API together with an event message we can have Misty say which of her touch sensors was triggered.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events

misty = Robot()

def touched(data):
    print(data)
    misty.speak("Hello there, this is my " +  data["message"]["sensorPosition"], 1)

misty.register_event(event_name='touch', event_type=Events.TouchSensor,
callback_function=touched, keep_alive=True)
misty.keep_alive()
```

Up Next 👇

PAGE
Lesson 5: Expand awareness

Last updated 1 month ago

👀 Lesson 5: Expand awareness

Aim

The aim of this lesson is to learn how to make Misty more aware of her surroundings using event-based skills with face and object recognition, as well as audio localization. By the end of the lesson, you will be able to build more complex skills in Python for your robot character.

Face Recognition

Challenge 1: Remember me?

Now that you've explored how to make Misty more aware of her environment and respond to human touch in Python, you can also make her recognize faces using her face recognition capabilities just like in Blockly. If you've trained Misty to recognize your face in Lesson 6: Face recognition you don't need to repeat the process for Python, your FaceID is already in her memory.

The amazing thing about programming is that once you have learned a way to solve a problem if you you can use the same solution for similiar problems. So even for the face recognition event the structure of the code for the function and for the event is the same as for the bump and touch sensors events with one exception. Some event functions require you to first enable the service you are using, in this case we need to use the `misty.start_face_recognition` API call to enable the service for the face recognition event.

In order to distinguish faces in Python you can write the FaceID name in the "Label" parameter and build your conditions with sequences as shown in the example below. Give it a try!

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events
import time

misty = Robot()
misty.start_face_recognition()

def recognized(data):
    print(data)
    if data["message"]["label"] == 'Simone':
        misty.speak("Hi Simone!", 1)
        misty.play_audio("s_Awe.wav", 50)
        misty.transition_led(0, 255, 0, 255, 127, 0, "TransitOnce", 1000)
    for i in range(2):
        misty.move_arms(80, -80, 50, 50)
        time.sleep(1)
        misty.move_arms(80, 0, 50, 50)
        time.sleep(1)
    elif data["message"]["label"] == 'Denis':
        misty.speak("Hi Denis!", 1)
        misty.play_audio("s_Awe2.wav", 50)
        misty.transition_led(255, 0, 0, 0, 0, 255, "TransitOnce", 1000)
    for i in range(2):
        misty.move_arms(80, -80, 50, 50)
        time.sleep(1)
        misty.move_arms(80, 0, 50, 50)
        time.sleep(1)
    else :
        misty.change_led(0, 0, 0)

misty.register_event(event_name='face_recognition_event',
event_type=Events.FaceRecognition, callback_function=recognized, keep_alive=False)
misty.keep_alive()
```

Challenge 2: Recognize more people

If you want Misty say hello to everyone she already recognizes you can generalize the code using event messages that we covered in the previous lesson. This will not only compact your code, but open up a new realm of interactions that you can create. Try editing the previous code using the example below.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events
import time

misty = Robot()
misty.start_face_recognition()
def recognized(data):
    print(data)
    misty.speak("Hi " + data["message"]["label"], 1)
    if data["message"]["label"] == 'Simone':
        time.sleep(1)
        misty.play_audio("s_Awe.wav", 50)
        misty.transition_led(0, 255, 0, 255, 127, 0, "TransitOnce", 1000)
    elif data["message"]["label"] == 'Denis':
        time.sleep(1)
        misty.play_audio("s_Awe2.wav", 50)
        misty.transition_led(255, 0, 0, 0, 0, 255, "TransitOnce", 1000)
    else :
        misty.change_led(0, 0, 0)
    for i in range(2):
        misty.move_arms(80, -80, 50, 50)
        time.sleep(1)
        misty.move_arms(80, 0, 50, 50)
        time.sleep(1)

misty.register_event(event_name='face_recognition_event',
event_type=Events.FaceRecognition, callback_function=recognized, keep_alive=False)
misty.keep_alive()
```

Object Detection

Challenge 3: What's that over there?

Well done so far! You can now make Misty recognize you and your friends, this is an essential element for robots to be able to establish and remember relationships just like us humans. But it doesn't stop there, you can even have her recognize objects like a TV, Toothbrush, Bicycle and much more. See Known objectsfor more information. Using the same event skill structure as for face recognition and the example below, try building a skill where Misty tells you what object she sees.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events
import time

misty = Robot()
misty.start_object_detector()
def recognized(data):
    print(data)
    misty.speak("Oh sweet, I think I see a" + data["message"]["description"], 1)
    if data["message"]["description"] == 'person':
        time.sleep(2)
        misty.play_audio("s_Awe.wav", 20)
        time.sleep(1)
        misty.speak("I love humans, they are my best friends")
        misty.transition_led(0, 255, 0, 255, 255, 0, "TransitOnce", 1000)
    elif data["message"]["description"] == 'tv':
        misty.speak("Lets watch some sci-fi robot movies together")
        misty.transition_led(255, 0, 0, 255, 127, 0, "TransitOnce", 1000)

misty.register_event(event_name='object_detection_event',
event_type=Events.ObjectDetection, callback_function=recognized, keep_alive=False)
misty.keep_alive()
```

Audio Localization

Challenge 8: Marco Polo

Besides recognizing faces and objects, you can also teach Misty to understand where your voice or a sound is coming from using audio localization. This is set up like any other event using the source tracking event together with the `misty.start_recording_audio` API. The tricky part about setting up an audio localization event is using the audio data to tell Misty in which direction she should move her head. To solve this you can set a variable called `doa` (degreeofarrival). Since Misty will create an audio recording, you will need to delete it once the event is complete. To do this, you can set up sensor event with one of her bumpers. Let's try playing the game Marco Polo using audio localization together with a face recognition event!

```
from mistyPy.Events import Events
from mistyPy.Robot import Robot

misty = Robot()
audio_file_name = "deleteme.wav"

def bumper_press(event):
    if(event["message"]["sensorId"] == "bfr" and event["message"]["isContacted"] ==
True):
        misty.stop_recording_audio()
        misty.delete_audio(audio_file_name)
        misty.unregister_all_events()
        print("finished")

def source_tracking(event):
    doa = event["message"]["degreeOfArrivalSpeech"]
    # Calculate the head angle based on the DOA (adjust the scaling factor as needed)
    head_angle = 90 - doa  # Subtract 90 degrees to center the head
    # Set the head movement duration and units as needed
    movement_duration = 1.0  # You can adjust the duration as needed
    movement_units = "degrees"
    # Move the head towards the sound source
    misty.move_head(0, 0, head_angle, duration=movement_duration, units=movement_units)
    print(f"DOA: {doa}, Head Angle: {head_angle} degrees")

def recognized(data):
    print(data)
    if data["message"]["label"] == 'yourname':
        misty.speak("Polo")
        misty.transition_led(0, 255, 0, 255, 127, 0, "TransitOnce", 1000)

# Register for the FaceRecognition event
misty.register_event(event_name='face_recognition_event',
event_type=Events.FaceRecognition, callback_function=recognized, keep_alive=True)

misty.register_event(event_type=Events.BumpSensor, event_name="bump pressed",
keep_alive=True, callback_function=bumper_press)
misty.register_event(event_type=Events.SourceTrackDataMessage, event_name="some name",
keep_alive=True, callback_function=source_tracking, debounce=1000)
misty.start_recording_audio(audio_file_name)
misty.start_face_recognition()
misty.keep_alive()
```

Up Next 🐥

PAGE
Lesson 6: Compact code

Last updated 2 months ago

🔗 Lesson 6: Compact code

Aim

The aim of this lesson is to learn how to use variables and functions in Python. While you've already learned how to use some functions, here you will be able to expand your capabilities with variables to create more readable, expandable and efficient code.

Variables

Challenge 1 : Control speed with variables

While getting into variables in Python may be scary at first, they are actually quite simple to use. You may recall from Lesson 7: Variables and Functionsthat they are a fundamental concept in programming that allow you to store and manage data throughout your program. They are containers that hold values, and you can refer to the values by using names.

A value in a variable can be both a number and a string. If you put the double quotes " " or single ' ' it will be identified as a string.

```
x = 50
y = "Hello"
```

You can use variables to simplify your code and make quick changes. Let's try rebuilding the variable blocks from Blocky in Python to control Misty's movement speed.

```
from mistyPy.Robot import Robot
import time

misty = Robot()
x = 40
speed = 60

misty.move_head(0, 0, 0, speed)
misty.move_arms(0, x, speed, speed)
time.sleep(1)
misty.move_arms(x, 0, speed, speed)
time.sleep(1)
misty.move_arms(0, x, speed, speed)
time.sleep(1)
misty.move_arms(x, 0, speed, speed)
time.sleep(1)
```

Remember to insert valid values in each comma space in a function. For example, a string where `is` expected an `int` is not accepted.

Challenge 2: Let's go global

Variables can be both defined inside a function (local variables) or made global. A global variable in programming is a variable that is accessible from any part of the program, unlike local variables which are accessible only within the scope they are declared as in the example above. Global variables are useful for storing data that needs to be accessed by multiple functions or parts of a program and can help you build larger and more complex event skill trees that use bump and touch sensors, face recognition, object detection, among others.

When you're using the same variable in multiple functions you will need to define it as a global variable and define event handlers using the following syntax:

```
#first function that defines the conditions of an event
def eventname_1(data):
global variablename
    print(data)
    if statement
    variablename= True
    elif statement
    variablename= False


#second function that defines the outcomes of an event based on the conditions set in
the first function
def eventname_2(data):
    print(data)
    if globalvariablename == True:
    #misty.display_image("e_Joy2.jpg", 1)
    else:
    #misty.display_image("e_Sadness.jpg", 1)
```

As you can see event handlers contain the code that defines what commands Misty should execute when a specific event is trigger. For example we can define our first event function as `eventname_1` to handle a group of bump sensor events. The second event function can be defined as `eventname_2`, which will handle the responses for the events that are triggered in the first function. Both functions are linked together by a global variable called `bumped`. The next step is to determine how you want the global variable to handle reponses to a triggered event. You can do this by using conditions `True` or `False` for your global variable. This way you can create multiple responses based on whether one of the conditions is met. Now let's try setting up an event skill with global variables using the example below.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events
import time

misty = Robot()

misty.display_image("e_Contempt.jpg", 1)
misty.change_led(255, 255, 255)
time.sleep(1)

def bumped_1(data):
    global bumped #global variable
    print(data)
    if data["message"]["sensorId"] == 'bfr':
        misty.change_led(255, 255, 0)
        bumped = True #using global variable
        misty.transition_led(255, 255, 0, 255, 255, 255, "TransitOnce", 1000)
    elif data["message"]["sensorId"] == 'bfl':
        misty.change_led(0, 100, 255)
        bumped = False
        misty.transition_led(0, 100, 255, 255, 255, 255, "TransitOnce", 1000)
    else:
        misty.change_led(255, 255, 255)

def bumped_2(data):
    print(data)
    if bumped == True:
        misty.transition_led(255, 255, 255, 0, 255, 0, "TransitOnce", 1000)
        misty.display_image("e_Joy2.jpg", 1)
    else:
        misty.transition_led(255, 255, 255, 255, 0, 0, "TransitOnce", 1000)
        misty.display_image("e_Sadness.jpg", 1)

misty.register_event(event_name='bumped_1', event_type=Events.BumpSensor,
callback_function=bumped_1, keep_alive=False)
misty.register_event(event_name='bumped_2', event_type=Events.BumpSensor,
callback_function=bumped_2, keep_alive=True)

misty.keep_alive()
```

Functions

Challenge 3: Make a call to your functions

As you discovered throughout your programming journey, functions are fundamental to building any complex event skill both in Blockly and in Python. Since they usually require a lot of space you will eventually find yourself going back and forth between lines of code to find that perfect sequence that you really want to use again in another part of your program. To avoid let's build up a list of functions that you can call anywhere in your code.

In Python a function is defined using the `def` keyword followed by a custom name. For example you can define your function as `Greeting`, `Driving`, `Dancing` and so on. If the function has no arguments you can just open and close your parenthesis "()", otherwise if you have arguments you can insert their name in the parenthesis and separate them with a comma " , ".

To call a function in your code you just need to write in it's name followed by parenthesis as shown in the example below. There isn't a specific order for defining the functions. You can define them both before and after your working code.

```
from mistyPy.Robot import Robot
import time

misty = Robot()

#functions local library
def Misty_waves(): #creating a function
    for i in range(2):
        misty.move_arms(0, -40, 50, 50)
        time.sleep(1)
        misty.move_arms(-40, 0, 50, 50)

def Hello_world():
    misty.speak("Hello world!", 1)
    misty.move_head(0, 0, 0, 50)
    misty.move_arms(0, -40, 50, 50)
    misty.display_image("e_Joy.jpg", 1)
    misty.transition_led(255, 255, 255, 0, 255, 0, "TransitOnce", 1000)
# end of functions local library

#code
misty.drive_time(30, 0, 5000)
Hello_world()#calls your function
time.sleep(1)
Misty_waves() #calls your function
#end of the code
```

Challenge 4: Combine variables and functions

Now that you've covered all the basic of using variables and functions, let's combine them to create a slim and neat event skill that can call various functions when an event is triggered.

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events
import time

misty = Robot()
misty.start_face_recognition()

def Misty_waves():
    for i in range(2):
        misty.move_arms(0, -40, 50, 50)
        time.sleep(1)
        misty.move_arms(-40, 0, 50, 50)

def reaction_to_Simone():
    #misty.play_audio("s_Awe.wav", 50)
    misty.transition_led(0, 255, 0, 255, 127, 0, "TransitOnce", 1000)

def reaction_to_Denis():
    #misty.play_audio("s_Awe2.wav", 50)
    misty.transition_led(255, 0, 0, 0, 0, 255, "TransitOnce", 1000)

def reaction_to_new_person(item):
    misty.speak("I'm Misty, nice to meet you" + item + "Why don't you come here to
memorize your face?" + "So next time I can recognize you and we can play!", 1)
    misty.display_image("e_Joy.jpg", 1)
    misty.transition_led(70, 100, 160, 127, 255, 0, "TransitOnce", 1000)

def recognized(data):
    print(data)
    global item #definition variable
    item = data["message"]["label"] #assigning a value to a variable
    misty.speak("Hi " + data["message"]["label"] + "Great to see you again", 1)
    if data["message"]["label"] == 'Simone':
        reaction_to_Simone() #calls your function
    elif data["message"]["PersonName"] == 'Denis':
        reaction_to_Denis()
    else :
        misty.change_led(0, 0, 0)
    Misty_waves()
    reaction_to_new_person(item)

misty.register_event(event_name='face_recognition_event',
event_type=Events.FaceRecognition, callback_function=recognized, keep_alive=False)

misty.keep_alive()
```

Up Next 👇

PAGE
Lesson 7: Start a conversation

Last updated 2 months ago

🗣️ Lesson 7: Start a conversation

Aim

In this lessons you will learn how to work with Natural Language Processing (NLP) concepts and Speech Recognition in Python and create your own conversations with Misty. By the end of lesson you will be able to master the full range of Misty's speech and conversational abilities.

Waking up

Challenge 1: Misty Wake-up

You may recall from the Blockly Lessons that before you start a conversation with Misty you can let her know that you want to get her attention with her default wake-up phrase "Hey,Misty". To do this, you will need to set up the start key-phrase recognition event. The code structure of the key-phrase recognized event is identical to any other event.

Once you have imported the libraries required, you can call the KeyPhraseRecognized event by building a callback function that will determine what you want Misty to do when the key-phrase is recognized. How would you choose to design your wake up sequence?

```
from mistyPy.Robot import Robot
from mistyPy.Events import Events
import time

misty = Robot()

misty.display_image("e_SleepingZZZ.jpg", 1)
misty.play_audio("e_Sleepy.wav", 50)
misty.move_head(60, 0, 0, 80)
misty.move_arms(85, 85, 80, 80)
misty.change_led(0, 0, 255)
misty.start_key_phrase_recognition()

def Key_Phrase_Recognized(data):
    print(data)
    misty.display_image("e_Surprise.jpg", 1)
    misty.play_audio("s_PhraseHello.wav", 50)
    time.sleep(2)
    misty.speak("Good morning")
    misty.move_head(0, 0, 0, 80)
    misty.move_arms(-85, -85, 80, 80)

misty.register_event(event_name='key_phrase_recognized',
event_type=Events.KeyPhraseRecognized, callback_function=Key_Phrase_Recognized,
keep_alive=True)
misty.keep_alive()
```

Building conversations and context

Challenge 2: Questions and Answers

Now that you've created your wake-up sequence in Python you can start building a conversation. In Blockly Lesson 8: NLP we used the conversation tree concept to understand the relationships between different elements of the conversation such as context, intents, sample, etc. In Python you can the same concepts to build your conversation. Now let's try rebuilding the YesNoQuestion conversation tree example from Blockly in Python.

The first step to build a conversation tree in Python is to think about how you want to structure your conversation tree using the `misty.create_state` API call, which will determine what Misty will say and do when an intent is triggered.

Each conversation state has some properties like:

- state name: The unique name for the state being created.
- speak: what you want Misty to during the state
- contexts: if you want Misty to recognize words within the context of the conversation you need to define it in the state. In the example below you can use "furhat.context.en-yes-no". As you may remember, a context in a conversation tree contains all the topics and words associated with them These are commonly referred to as 'intents' and 'samples'.
- startAction, you can decide which action Misty will perform in that specific conversation state, you can find the list of Misty's pre-built actions here NLP Actions

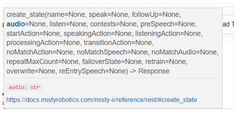
```
from mistyPy.Robot import Robot

misty = Robot()

# Define the conversation states
misty.create_state(name="start", speak="Do you like robots?",
contexts="furhat.context.en-yes-no", startAction="listen")
misty.create_state(name="HandleYes", speak="Yay, that's awesome!")
misty.create_state(name="HandleNo", speak="Wow, it's so obvious that humans will
eventually be replaced by robots.")
```

Important: If you want Misty to listen to you speaking during a state, it's important to select 'listen' in the action field. This way Misty will recognize your speech and reference it with the intents.

There are many more parameters you can modify while creating a conversation state:



Once you've created your states you will need to determine how you want Misty to transition between one state and another. To achieve this you can use the `misty.map_state` API call.

In the map state, it's necessary to insert the name of your conversation, the initial state and the next state. To let Misty know which words should trigger a transition between states you can use triggers.

Important: The trigger name should match the intent name.

In Misty's default context furhat.context.en-yes-no for example you can find two intents to trigger: "yes" and "no". After defining the structure and flow of the conversation you can start your conversation using `misty.start_coversation`

```
from mistyPy.Robot import Robot

misty = Robot()

# Define the conversation states
misty.create_state(name="start", speak="Do you like robots?",
contexts="furhat.context.en-yes-no", startAction="listen")
misty.create_state(name="HandleYes", speak="Yay, that's awesome!")
misty.create_state(name="HandleNo", speak="Wow, it's so obvious that humans will
eventually be replaced by robots.")

# Define the conversation transitions
misty.map_state(conversation="YesNoQuestion", state="start", trigger="yes",
nextState="HandleYes")
misty.map_state(conversation="YesNoQuestion", state="start", trigger="no",
nextState="HandleYes")

# Start the conversation
misty.start_conversation("YesNoQuestion")
```

Challenge 3: Build your own conversation

Awesome job! Now that you know how to build a conversation tree in Python, let's learn how to build a conversation from scratch using your own context with intents and samples.

As you may remember from Blockly, the first step is to build up your context. A context is composed of two or more intents, each intent has a name and some samples. To be able to set up intents in your context you will need to class them as a list and group them in your context using a variable called `blob`. Once you've created all these elements you can use Misty's command `misty.train_nlp_engine` to have her store the context in her memory.

```
from mistyPy.Robot import Robot
import json
from json import JSONEncoder

misty = Robot()

class IntentEncoder(JSONEncoder):
        def default(self, o):
            return o.__dict__

class Intent:
    def __init__(self, name, samples):
        self.name = name
        self.samples_list = samples

pizza_list = ["pizza", "pizza pie", "slice of pizza"]
taco_list = ["taco", "tacos", "mexican food"]

i1 = Intent("pizza", pizza_list)
i2 = Intent("taco", taco_list)

blob = json.dumps([i1, i2], indent=4, cls=IntentEncoder)

misty.train_nlp_engine(context="sample__food", intents=blob, save=True, overwrite=True)
```

There's plenty of action in Misty's storage, but if you want to create your actions you can use the `misty.create_action` API. In the parameters you will need to define the unique name of the action and a script, which contains all of the actions you want Misty to perform.

```
misty.create_action(name="question_action", script= "LED-
PATTERN:0,0,255,40,0,112,1200,breathe;IMAGE:e_ApprehensionConcerned.jpg;ARMS:29,29,1000;
HEAD:10,0,0,1000;", overwrite= True)
```

Once you have created your context and your actions, the only thing that's missing is setting up the conversation tree with states.

```
misty.create_state(name="FoodStart", speak="Do you prefer tacos or pizza",
contexts="sample__food")
misty.create_state(name="TacoState", speak="Yeah, tacos are yummy, but pizza is way
better.")
misty.create_state(name="PizzaState", speak="I will fight anyone who doesn't think pizza
is the alpha food!")

misty.map_state(conversation="FoodConversation", state="Foodstart", trigger="taco",
nextState="TacoState")
misty.map_state(conversation="FoodConversation", state="Foodstart", trigger="pizza",
nextState="PizzaState")

misty.start_conversation("FoodConversation")

misty.keep_alive()
```

Congratulations on completing your own conversation tree! You are ready to combine all that you've learned so far to create amazing human-robot interactions using event skills together with physical touch, face recognition, object recognition, audio localization, speech recognition and natural language processing.

Coming Soon 👉 Lesson 9: Connecting External APIs

Last updated 1 month ago