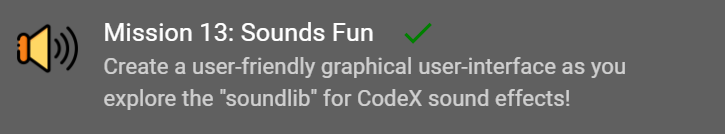
**Mission 13:**

**Sounds Fun**

**Student Workbook**



****

**Let’s get animated**

For this mission, you will learn how to play sounds and music in the background while other code is running, make sound effects for games and user feedback, and control the pitch and loop sounds.

Go to the Mission 13 Log and fill out the   
Pre-Mission preparation.

* In this mission you will use CodeX as a controller to start and end a race. With batteries, CodeX can go anywhere! What is something you might want the CodeX to do or control?

**Mission 13: Sounds Fun**



**Picking up good vibrations?**

Previously you played MP3 files on CodeX using the basic audio functions. But there is much more you can do with sound on this amazing device! In this mission you will dive deep into the soundlib module.

**Get GUI!**

You will also make a graphical user interface, or GUI. It will give the user a way to interact with CodeX.

**In this project you will make a race day controller that can:**

* Play sounds and music in the background while code is running
* Make sound effects
* Control the pitch
* Loop the sounds
* Craft a GUI component

**Mission 13: Get started**

* Go to <https://make.firialabs.com/> and log in.



* Go to Mission 13



* Click and start Mission 13.

**Objective #1: Race Day**

A big cycling race starts in just a few hours. But the sound system is broken!   
If a solution isn’t found, the event will   
be canceled.

Noooo! You have to save the day!

Your CodeX has lots of sound capabilities. And, you can plug the output into a guitar amp to get the volume up.

**Controller Requirements**

Here are the requirements from the race officials. They made a napkin-sketch and gave it to you.

The controller (CodeX) must:

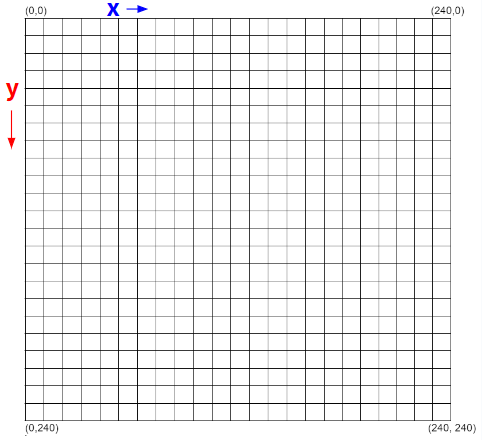
* Have an easy-to-use user-interface (UI)
* Loop the race theme music in the background (Play/Pause control)
* Trigger the “start” with a sound effect
* Trigger the “finish” with a sound effect
* Give a “warning” siren if needed

**Objective #1: Race Day**

**Graphical User Interface (GUI)**

You will use text and boxes to design your screen.

Think of the screen as one quadrant on a graph.



* The origin is in the upper left corner
* The **x** values go across, to the right
* The **y** values go down, to the bottom
* All **x** and **y** values are positive

**Objective #1: Race Day**

**Graphical User Interface (GUI)**

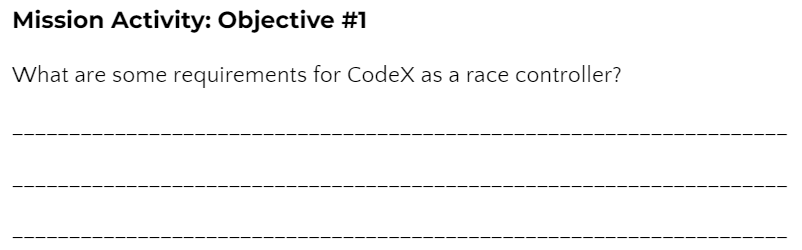
Use these commands to draw shapes.

**x** and **y** represent the position on the graph   
(like an ordered pair).

* display.fill\_rect(**x**, **y**, width, height, color) &   
  display.draw\_text(“text”, scale, color, **x**, **y**)
  + For the controller title at the top
* display.draw\_rect(**x**, **y**, width, height, color)
  + Box for each task (4 total)
* display.draw\_text(“text”, scale, color, **x**, **y**)
  + Text for each task (4 total)
* display.fill\_rect(**x**, **y**, width, height, color)
  + Filled box to indicate the task selection

**DO THIS:**

Go to your Mission Log and write the requirements for CodeX as a race controller.

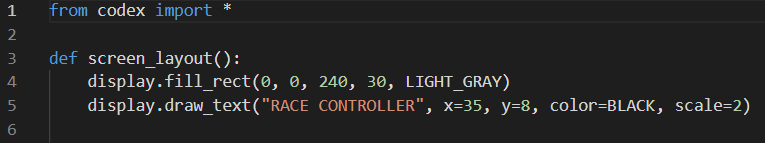


**Objective #1: Race Day**

**DO THIS:**

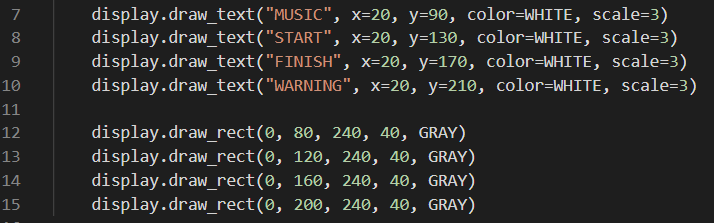
* Start a new file named **Race\_Control**
* Follow the CodeTrek, or the instructions below, to:
  + Define a function for the screen layout
  + Create a heading for the controller
  + Draw rectangles and text for the menu options

**Step 1 and 2**

* Define a function **screen\_layout()**
* Draw a filled rectangle and text for the UI heading

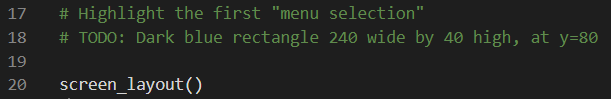
**DO THIS: Step 3**

* Complete the function by drawing text and rectangles for the menu options



**Step 4**

* AFTER the function, draw the dark blue filled rectangle that will show the selected menu option.
* Then call the function.



**Objective #2: Getting Interactive**

Your UI layout looks great! Now it’s time to scroll the buttons so the user can select different options.

* Right now the selection is stuck on MUSIC

Remember in Personal Billboard Mission 7 you:

* used a list to hold different items
* scrolled through the list
* used an index variable “choice”

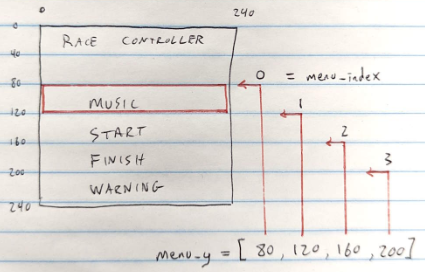
You can do the same thing here!



**Objective #2: Getting Interactive**

This time your list will hold the y-coordinates where each rectangle needs to be drawn.

* The list is **menu\_y**
* The variable **menu\_index** points to the selection.
* The x-coordinate is always **0**

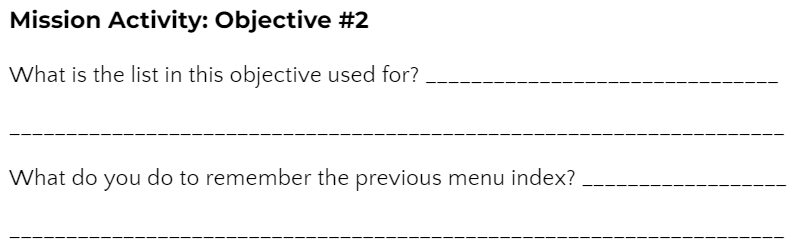


You will add code like this:



**Objective #2: Getting Interactive**

**DO THIS:**

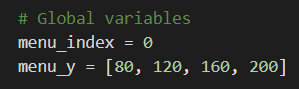
* Go to your Mission Log and write the answers to the questions.

Follow the CodeTrek or instructions here

**DO THIS:**

After the **screen\_layout()** function

* Add the global variable **menu\_index**
* Add the **menu\_y** list

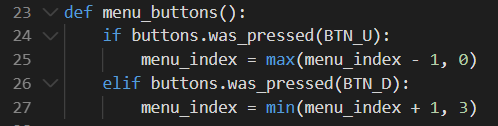


**Objective #2: Getting Interactive**

**DO THIS:**

Just below the function **screen\_layout()**

* Define a function **menu\_buttons()**
* It will check if **BTN\_U** or **BTN\_D** was pressed and update **menu\_index**

****

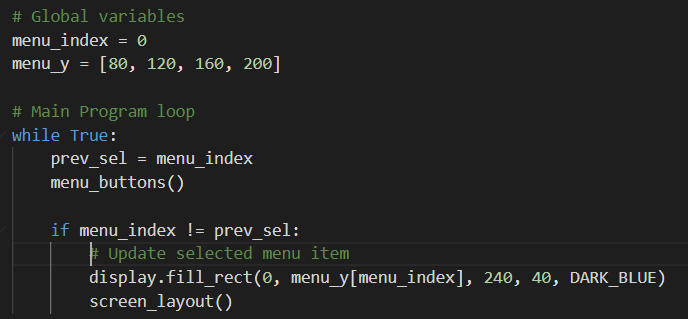
**NOTE:** *“max()” and “min()” keep the index at 0, 1, 2 or 3 without using an if statement.*

**Objective #2: Getting Interactive**

**DO THIS:**

Move the rest of the code into a **while True:** loop

* Use a variable **prev\_sel** to keep track of the previous selection
* Call **menu\_buttons()**
* If the new **menu\_index** is different from the previous index:
  + Update the DARK\_BLUE rectangle
  + Call **screen\_layout()**



When you run the code, and press a button, you WILL get an ERROR!

****

**Objective #3: Bug Bashing**

So what’s up with this error message?



**Local Variables**

* When you assign (update) a variable inside a function, Python assumes it is local.
* A local variable is “private” to the function.
* It is a separate variable, even if it has the same name as another variable outside the function.
* It only exists while the function is running.
* It can’t hold its value between function calls.

**Global Variables**

* Exist for the entire life of the program
* Defined outside any function
* Can be used inside any function
* If the variable changes (gets updated), you must use a **global** statement

**Objective #3: Bug Bashing**

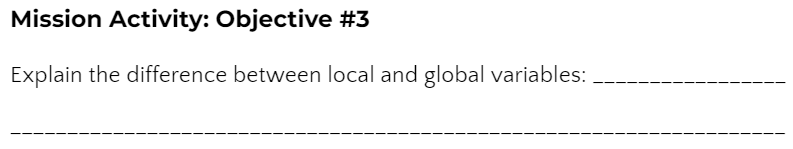
So what happened?

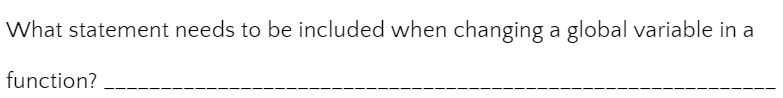
* The **menu\_buttons()** function changed the value of **menu\_index**
* But Python thinks **menu\_index** is a local variable!
* And a local variable doesn’t exist until it is first assigned a value in the function
* But the max() and min() functions tried to read the value of **menu\_index** before one was given

Instead, make **menu\_index** global by using the global statement.

**Objective #3: Bug Bashing**

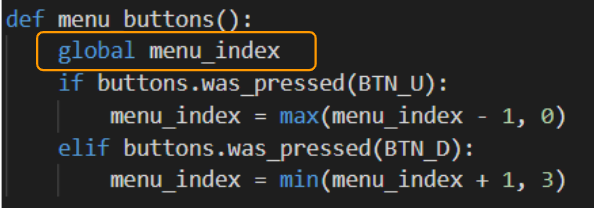
**DO THIS:**

* Go to your Mission Log and write the answers to the questions.



**Objective #3: Bug Bashing**

**DO THIS:**

* Add the **global** statement to the function to keep **menu\_index** global and not local.

**Note:** *The menu will still not work as expected. We will fix that!*



**Mission Quiz: Globals and Locals**

Test your skills by **taking the quiz**.

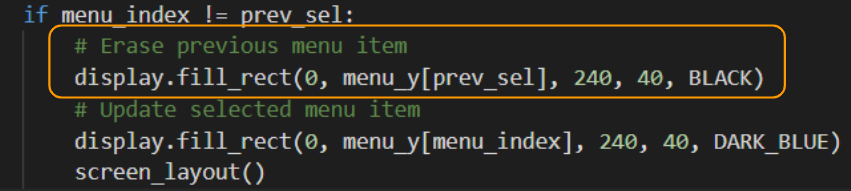
**Objective #4: Covering Your Tracks**

Oh dear! Your program has yet another problem. Your menu selection is leaving big footprints behind!

* You are keeping track of your previous selection (**prev\_sel**)
* Use it to “erase” the rectangle and then redraw it around the new selection (**menu\_index**)

**DO THIS:**

* Draw a BLACK rectangle at **prev\_sel** to “erase” the previous menu selection.

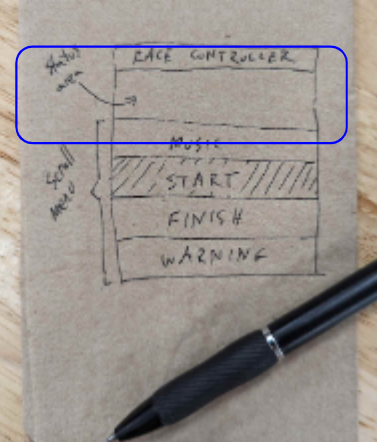


**Objective #5: Add Some Action!**

Now that your menu-scrolling is on point, it’s time to add a way for the user to trigger the selected action.

* For now just display a message when **BTN\_A** is pressed.

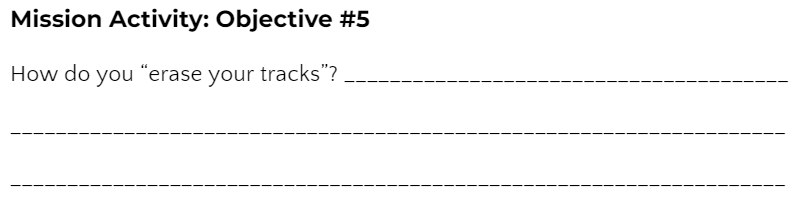
If you look closely at the napkin sketch there was a “status area” just above the menu.

* Put some text there for each menu action
* Soon you will add the sounds!

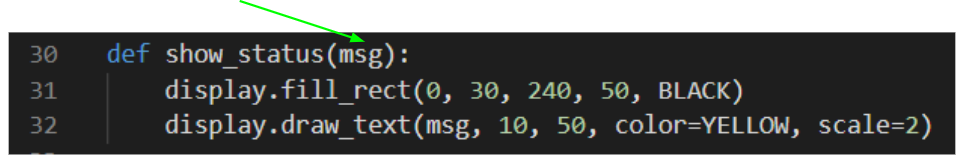
**Objective #5: Add Some Action!**

**DO THIS:**

* Go to your Mission Log and write the answers to the questions.



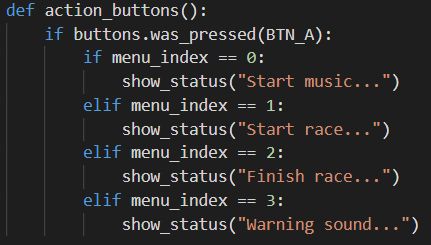
**DO THIS:**

* Define a function that shows a message for the selection.
* Use a parameter for the message.

**Objective #5: Add Some Action!**

**DO THIS:**

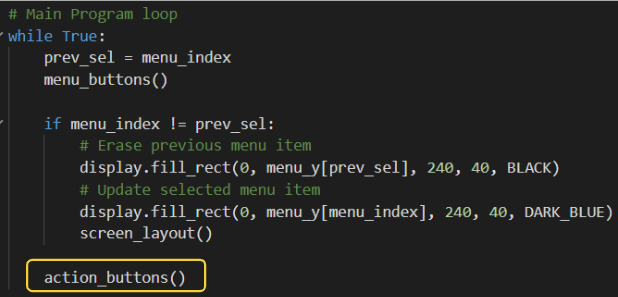
* Define another function that calls **show\_status()** and passes in the message to display.
* The variable **menu\_index** points to the items in the menu



**Objective #5: Add Some Action!**

**DO THIS:**

* Call **action\_buttons()** in the **while True:** loop (but not the **if** statement)
* Be careful with your indenting!

****

**Objective #6: Make a Good First Impression**

The menu system is looking good, but it could be more user-friendly.

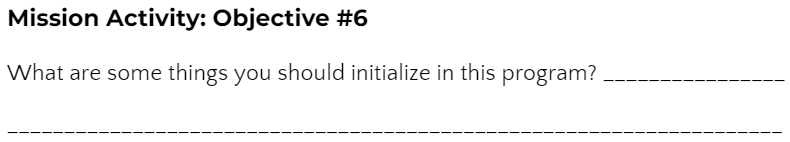
* The user has to press **BTN\_U** or **BTN\_D** before anything is displayed.
* There are no instructions about what buttons to press.

You should have a ***program initialization*** that sets the display and global variables when the code is first run.

* Use a global Boolean variable to indicate initialization
  + **init = True**
* Call **screen\_layout()** to draw the whole screen plus menu selection when the program runs

**DO THIS:**

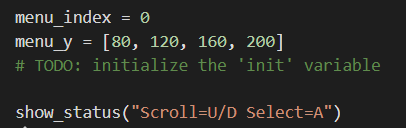
* Go to your Mission Log and write the answers to the question.



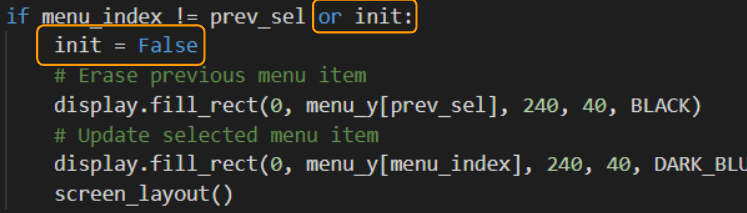
**Objective #6: Make a Good First Impression**

**DO THIS:**

* Initialize the Boolean **init** variable
* Call **show\_status(“Scroll=U/D Select=A”)**

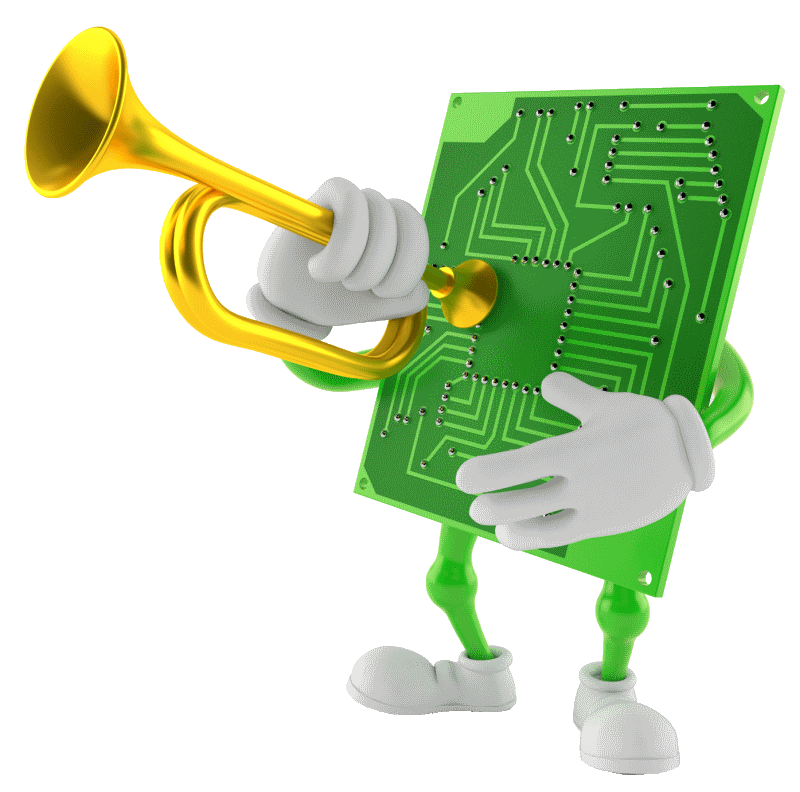
**Note:** *This is* **before** *the main loop, so it only runs once when the program first starts.*

* Modify the **if** statement in the loop to also check **init**
* Then update **init** to False inside the **if** statement
  + You only need it to be True for the first, or initial, run

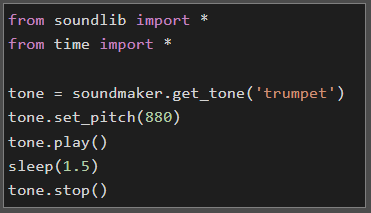
****

**Objective #7: START!**

Now it is time to put some ACTION in your menu actions!

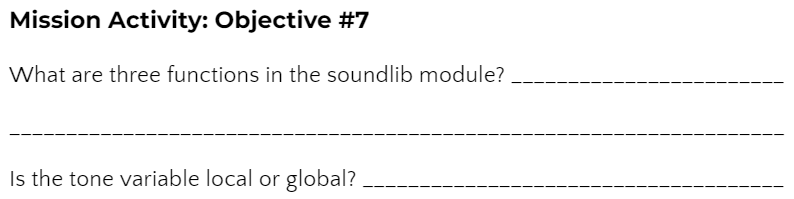
* To start the race you want a punchy distinctive sound.
* Something like a bugle call.
* Instead of playing an MP3 file, create your own tones and change the pitch (frequency) with code!

Your CodeX has an awesome Python module (or library) for creating music and sound effects.

* Import the **soundlib** module.
* Use **soundmaker** to get a tone or play an MP3 file.

**Objective #7: START!**

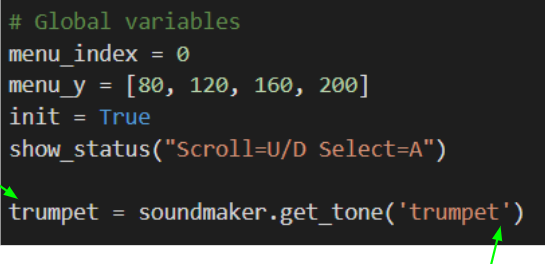
**DO THIS:**

Go to your Mission Log and write the answers to the questions.

**DO THIS:**

Define a global variable **trumpet** for the tone.

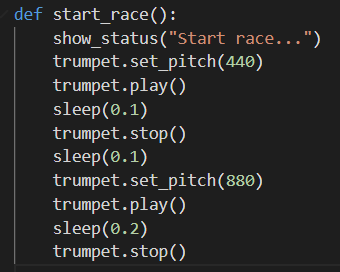
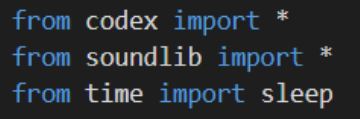
* Each call to **get\_tone()** gets a new sound that can be played independently.
* You can have up to 16 tones playing at the same time!

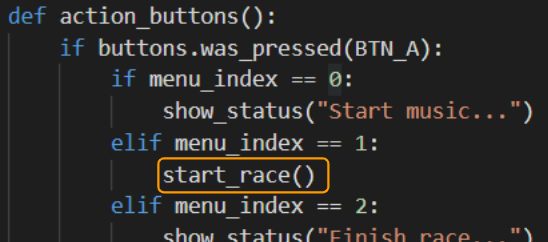


**NOTE:** You can use either double quotes or sing quotes for a string.

**Objective #7: START!**

**DO THIS:**

* Define a function to start the race.
  + Keep the sound effect simple.
  + You can customize the sound later.
* Remember your imports!

* Call the **start\_race()** function in **action\_buttons()** for **menu\_index == 1**

**Objective #8: More Fanfare!**

Now add some excitement to the fanfare starting sound.

* Repeating the first tone several times is a good lead-in
* And you know how to do that without copying and pasting a bunch of times, right?!

You need to use a loop!

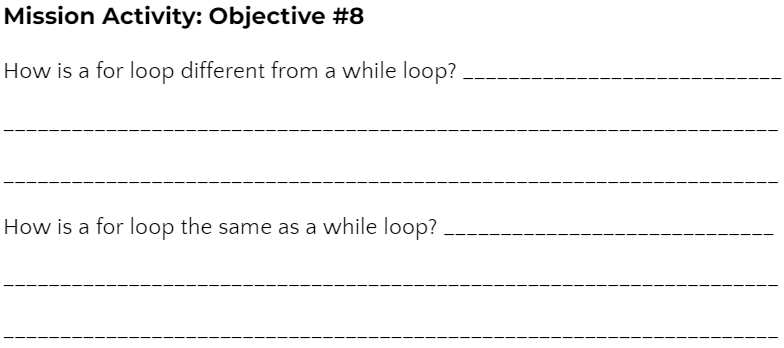
* But this time you can make the code even more simpler…

The **for** loop is made for looping across a range of numbers, or iterating over a list.

* Use the built-in **range()** function to specify the numbers to loop across.
* The **for** loop saves you the trouble of initializing and updating the loop control variable.
* This is because it automatically increments the control variable.

**DO THIS:**

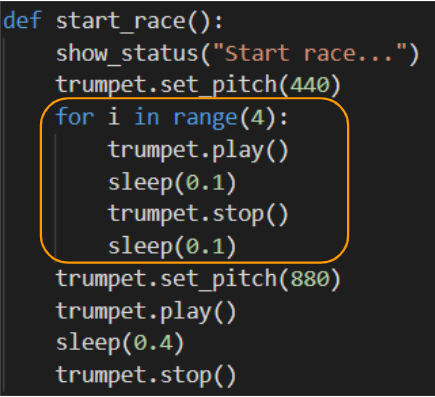
Go to your Mission Log and write the answers to the questions.



**Objective #8: More Fanfare!**

**DO THIS:**

* Add a **for** loop to the **start\_race()** function.
* Play some sounds in the function.
  + Try it as shown,
  + Then feel free to get creative!
  + You can even add another for loop.

****



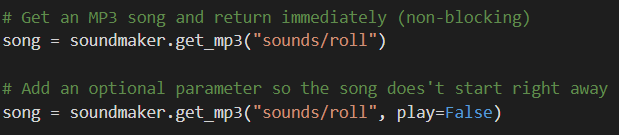
**Mission Quiz: For Loops**

Test your skills by **taking the quiz**.

**Objective #9: Make Some Music!**

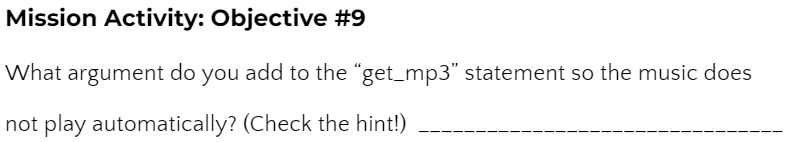
Now write code for the MUSIC menu option.

**Playing an MP3 in the background**

* With the soundlib module, you have a new way to play MP3 files.
* It is a non-blocking function, so your code can continue executing commands while the sound keeps playing.
* The **audio.mp3()** command you used in Mission 5 is a blocking function, and no code was executed while the file played.
* The new command to use is:



**DO THIS:**

Go to your Mission Log and write the answers to the question.

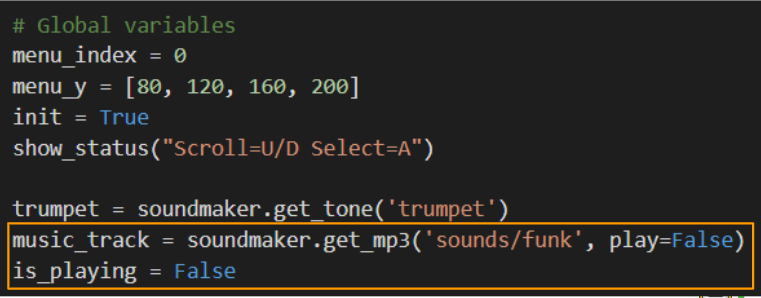
**Objective #9: Toggling On/Off**

When the user selects MUSIC, it should either turn the music ON or OFF.

* This depends on whether the music is playing (True or False).
* So you can use a Boolean variable and keep track.
* When the MUSIC button is pressed, the state can flip:
  + True -> False or False -> True
* That is exactly what the not logical operator does.
* You will often use this operator when you need to toggle   
  (or flip) a Boolean value.



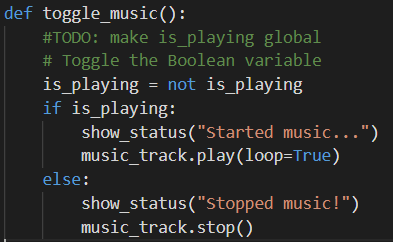
**DO THIS:**

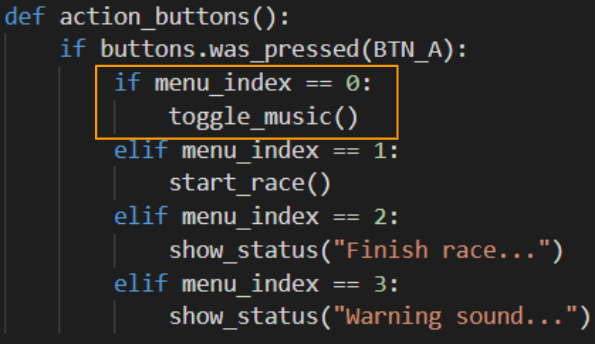
Add two global variables:  
One for the MP3 file and one for the Boolean toggle variable

**Objective #9: Make Some Music!**

**DO THIS:**

* Define a function to play the music.
* Use the Boolean toggle variable to start and stop the music.



* Call the toggle\_music() function when MUSIC is selected.



**Mission Quiz: Blocking and Toggling**

Test your skills by **taking the quiz**.

**Objective #10: The Sound of Victory**

It is time for you to craft another sound.

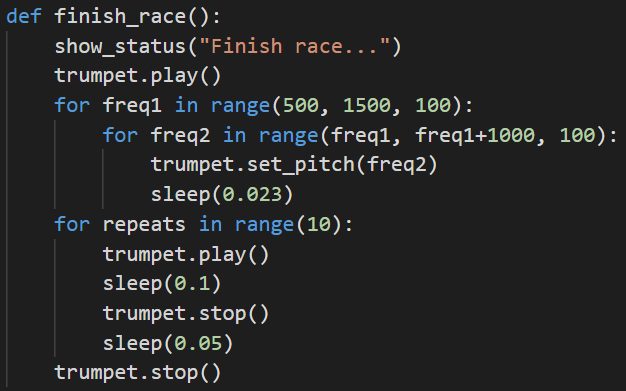
* The next one will be played when a racer crosses the finish line.
* It should be exciting and inspiring – a celebration of effort!

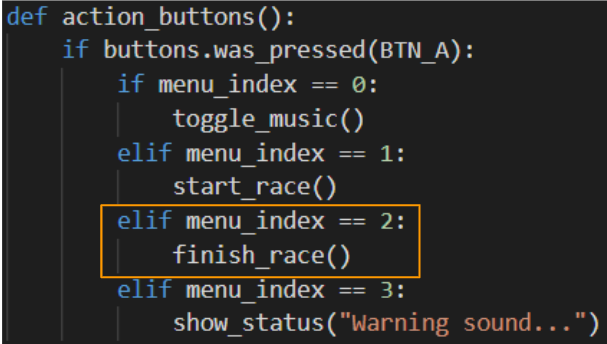
The START sound used a for loop to repeat a tone. Now try changing the frequency inside your loop using set\_pitch().

* *The CodeTrek will lead you to a pretty cool sound. But you can do better and* ***make it your own****.*

**Objective #10: The Sound of Victory**

**DO THIS:**

* Define a function to finish the race with a victory sound.
  + Use a for loop to change the frequency of the tone.
  + Use a for loop to repeat a tone several times.

* Call the function **finish\_race()** when FINISH is selected.

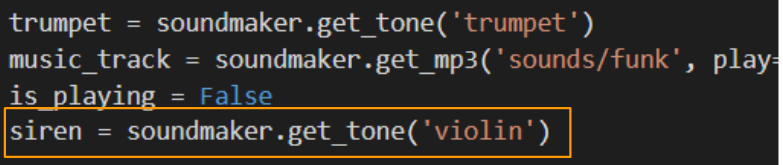
**Objective #11: Warning Alert**

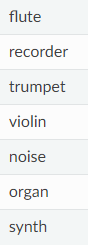
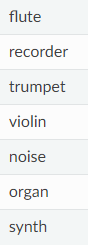
If there is a problem on the race course, or some danger to warn the riders about, how will the race officials get their attention?

* Create a sound effect that makes people take notice!
* Use the **soundlib** feature: **glide()**
* The **glide(new\_pitch, duration)** function is an easy way to ramp the pitch from the current pitch to a **new pitch** over a specified amount of time (**duration**).
* This is easier than using a loop, plus it is non-blocking.



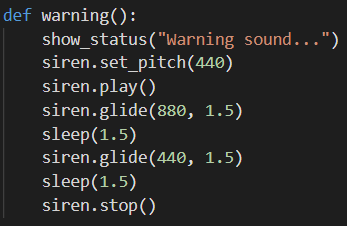
**DO THIS:**

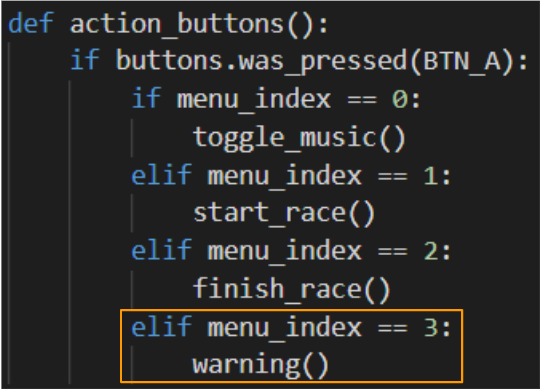
* Define a new tone **siren** for the sound effect.
  + You can select any tone from the list.



**Objective #11: Warning Alert**

**DO THIS:**

* Define a function for the warning.
  + Use the **glide()** function.
  + It takes two arguments:
  + **new\_pitch** is where it will end
  + **duration** is how long it takes to get there.
  + You still need to **sleep()** while it is gliding because it is non-blocking.

* Call the function warning() when WARNING is selected.

**Mission Complete**

You have completed the thirteenth mission. 

**Do this:**

* Read your “Completed Mission” message
* Complete your Mission 13 Log
  + Post-Mission Reflection
* Get ready for your next mission!

**Wait! Before you go … Clear the CodeX**

Go to FILE -- BROWSE FILES

Select the “**Clear**” file and open it

Run the program to clear the CodeX

**Okay. Now you can go.**