Student Robotics 2022 Microgames 🥫

Welcome to the Student Robotics 2022 microgames! You will be learning how to use our kit by completing a series of challenges.

Step 0 - Join our Discord

Our Discord forum is *the best* way to chat with other competitors and our volunteers. Our friendly volunteers are able to help you solve any problems you may have building your robot. (Our unfriendly volunteers aren't given Discord access).

How do you join our amazing community? You first need your **team password**, which was emailed to your supervisor. Once you have that ready, you can register and join the discord forum with the invite link below, be sure to use 1 account per person.

When you're in, first thoroughly read through the rules in the #welcome-and-rules channel, then you will need to send the password to the #welcome-<yourname> channel to get full access.

Task:

Join with the link above, then say **two true things** and **one lie** about you or your team in #say-hello.

Kit Section

This bit of the microgames requires a Student Robotics kit. You can do this later if you don't have yours quite yet.

Step 1 - Update your robots' microSD card 🔽

You have your robot kit with you, hooray! However, your kit will still be running old software from last year's competition. You do need to flash your SD card with the latest version of the robot software. We have some instructions on how to do this in our handy documentation site, at studentrobotics.org/docs.

The bit you'll want to follow is: the page for the brain board. Our documentation should be your go-to place to find information about our robots, so you should learn how to read and navigate it. Please ping us on Discord if you have any problems.

Step 2 - Battery Charging 🔋

The two Lithium Polymer (LiPo) batteries in your kit together store the same energy as a **hand grenade**.

We take battery safety very seriously, and so should you.
(or else)

You **MUST** know how to *safely* charge the batteries in your kit.

The best place to start if you don't know how to do something is always the docs, at studentrobotics.org/docs. In the docs, we have detailed instructions on how to use the battery chargers in your kit.

We have two different kinds of chargers, make sure you use the instructions for the one you have in your kit.

Task:

Connect a battery and start charging it in the battery bag. **Read the instructions above,** the most obvious way is not necessarily the correct way. Please get your supervisor to check that things are plugged in correctly before you start charging

Step 3 - Assemble your kit 🏗️

Now you know how to charge your battery, you can start plugging everything together.

You should start by assembling your kit and getting used to what's in the box. You need to get used to checking for information in the docs: studentrobotics.org/docs. It's very important to follow the instructions carefully.

> **Warning**: The white plug on your battery is only used for charging. **Do not try to plug it into anything in your kit. It will damage the battery and the kit irreparably.**

At our Assembly page in the docs, there's a video tutorial from a few years ago on how to put it all together. If you have any more questions, don't hesitate to ask in Discord.

> Note: Your kit won't turn on unless you have a jumper cable or switch plugged into the 'on|off' terminal! If there isn't already one in your kit, you can make a jumper cable by wiring both terminals of a 5mm camcon (medium-sized green connector) together, and plugging it into the slot next to the on|off button.

Take your time, once you're happy that you've assembled your kit, and you know all of the components so you can move on to the next task.

Step 4 - Hello Kit 👋🧰

Now you've got a shiny kit ready and working, it's time to run some code on it!

The purpose of this task is to make sure you've got your robot running code!

In order to put code on a robot, all you need is a computer that can open Zip files and a USB stick. You can follow the instructions on how to program the robot in the 'Getting Code on the robot' section of the docs.

Now you will need to write the code to put into your robot. Our robots are programmed in python. We strongly recommend you use a code editor to write your code. The editor which comes default with Python is IDLE, which you may have used before. There are many much better code editors around, so check out our Code Editors page in the docs to see what you could use.

As a start, the programming tradition is to write a "Hello world":

```
print("Hello world")
```

Once you're printing "Hello, World!", you can try and find the secret password, add the lines below and see what it prints. Don't worry, this code is meant to be nearly impossible to read, so you don't cheat.

```
from sr.robot3 import Robot as R
print(
    f"{hex(15)[2]}{'pile'[2::-1]}",
    f"{'nuance'[-2::-2]}{R(auto_start=True).motor_board.serial_number[0]}",
)
```

The code is also available at pastebin.com/PWGDpaTm if you can't copy/paste.

You can check with your supervisor on what the secret code is.

As an additional challenge, make the program print the same message to the log 10 times. **Hint**: Use a for loop!

Step 5 - Broken Code 💔

Engineers often come across many problems, anything from missing brackets in code or a cat jumping onto their keyboard — hopefully you'll only be dealing with the former.

To help prepare you for this, we would like to look at a piece of code, and point out any errors, and then fix them. Copy the code below (also found on pastebin.com/wuRwjN0c) into the IDE and look at the code. Fix it, and then run your code in the simulator, which can be found in the docs.

This can be a very tricky task for people who are new to Python, if you need help, first look in the docs at Python Troubleshooting page. If you have any other issues, ask us on Discord!

```
from sr.robot3 import *
import time

R = Robot()
```

```
def drive_forwards(speed):
    R.motor_board.motors[0.power = speed
    R.motor_board.motors[1].power = speed


def drive_backwards(speed):
    r.motor_boards.motors[0].power = -speed
    R.motor_boards.motors[1].power = -speed


def look_for_marker_3()
    marker_ids = R.camera.see_ids()
   arena_markers = []
    for marker in marker_ids:
        if marker.id = 3:
            arena_markers.append(m)
    return arena_markers

while True:
    markers = look_for_marker_3()
    if markers:
        Drive_forwards(0.2)
```

You can check your answers with your supervisor.

Step 6 - Lift off! 🚀

A useful feature of python is the ability to pause your code for a given amount of time.
To do this, we will use a function in the "`time`" module. We have a few links to tutorials and the official python documentation at [the Python page in the docs](#).

The power board has a small piezoelectric buzzer on it. You may be able to spot it, it's right next to the fan. It's primarily used for diagnostics (the beeps you hear when you switch the kit on indicates the version of the software it's running) but you can also program your robot to beep at a particular tone for a set length of time. In other words, your robot is a musical instrument! We have documentation on how to beep your buzzer in the [power board programming section](#).

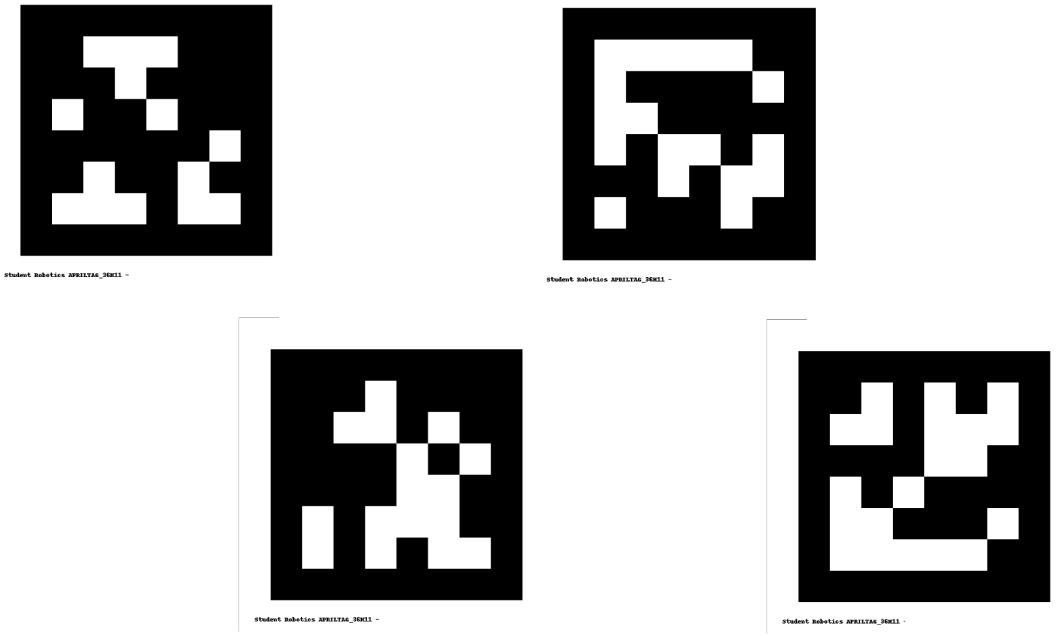
For this task, make a robot that starts a countdown by beeping 10 times with a one second delay between each beep, and then one long beep at the end. Once that's working, record a video of it and show it to your supervisor.

Step 7 - Look for things 👀

Your robot has a webcam! This webcam can spot our markers from far away, and our API gives you their distance in meters and both the angle of the face relative to the robot, and angle of the marker relative to the robot.

To get you used to writing code to use the webcam, we've prepared a challenge:

/\ This way up /

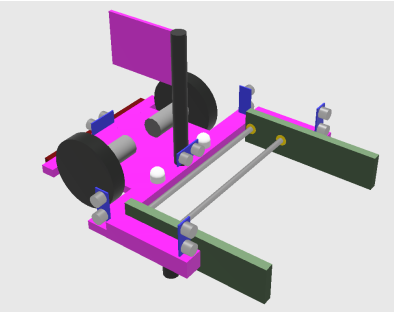


[Click here for a bigger version.]

Print out, or show the above image on your screen, try to convert the IDs from numbers into letters, it should make a word when read from Left to Right, if you use 0 is A, 1 is B, 2 is C, etc…

Our documentation on vision would be the place to start. Try to write a program that prints the entire answer in one go! That is, it prints the markers from left to right.

Simulator Section



We've prepared a basic robot simulator to help you learn to program your robot, this section uses that, rather than the physical kit, as you don't yet have all the motors and sensors hooked up. It's designed to be as similar to programming a real robot as possible, but there are some subtle differences.

Step 1 - Install the Simulator

First, you need to get the simulator working! Follow the instructions on the Simulator page in the docs. You will need to download both Webots and our simulation files, and also make sure you have python installed. Our documentation should be your go-to place to find information about our robots, so you should learn how to read and navigate it.

Run the test robot code. Once you've downloaded both files, install Webots, then extract the 'competition simulator' zip. This is going to be your main workspace for the simulator. Double click on the `arena.wbt` file (in the '`worlds`' folder) to run the simulation. It should launch webots (if not, ask for help!), and then start running a simulation of our sample robot. If you get an error, you'll need to make sure that python is installed correctly.

After running the simulation, you can check your folders, a new file named '`robot.py`' should have appeared just outside the 'competition-simulator' folder. This is our sample robot code (if a file ends in .py, it means it is a python file).

We strongly recommend you use a code editor to write your code. The editor which comes default with Python is IDLE, which you may have used before. There are many much better code editors around, so check out our Code Editors page in the docs to see what you could use.

Step 2 - Hello Simulator 👋🧑‍💻

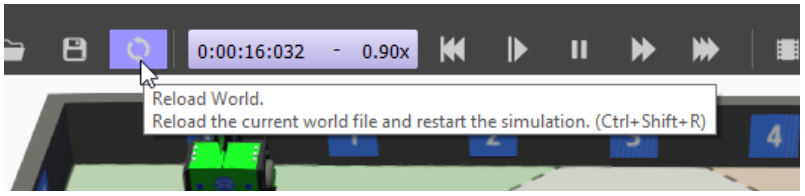
A tradition all programmers do when they're learning to program is to get their code to write the words "Hello World". In this challenge, we're making things a bit more complicated, instead of "Hello World" we're going to run some code to get a secret password.

Now, copy and paste the following code into the top of your `robot.py` (after you create your robot), which should be just outside the 'competition-simulator' folder:

```
print("Secret Code: H3{35%12}0 W{0}{chr(82)}LD")
```

(Don't worry if you don't understand it, we wrote it to be deliberately hard to understand, so you can't figure out what it does before you run it)

Then, after saving, you can run the code by either double-clicking on the Webots `arena.wbt` file (in the 'worlds' folder) , or by clicking 'Reload World' at the top of the screen.



The message should appear in the 'Console' at the bottom.

If you don't have a console at the bottom of the window, try resizing the main view of the simulation -- it may just be hidden. If that doesn't work, you can create the console by going to Tools > New Console.

If you get an error message, it might be that you have a version of python lower than our minimum, Python 3.9. You can install a later version at www.python.org/downloads/.

If you find the secret code, well done! You can check the answer with your supervisor, and move on to the next task.

Step 3 - Motor Movement 🛵

In order for your robot to move, you're going to need to know how to move the motors. The robot has 2 motors, one on each side of the robot.

You can set the motor power to anything between -1 and 1. To find python functions that change the robot itself, you should read our documentation, specifically the Programming category. The part related to motors will be on the motor board page.

Our simulation robot this year only has 1 motor board plugged in, you have two motor boards in your kit, meaning you can power up to 4 motors.

Now, make the robot drive forwards for 4 seconds, then drive backwards for 2 seconds. For the simulator, you should avoid using the normal python way to wait (`time.sleep()`), as it doesn't guarantee it will run the same on every computer. Instead you should use `Robot.sleep()`, like this:

```
R = Robot()
R.sleep(seconds)
```

Use this to make the robot wait for a few seconds. While the robot waits, its motors keep moving at the same speed you last set them to beforehand.

Step 4 - Servo Movement 🦾

In addition to motors, your robot can power servos. Servos are like motors, except they know how much they've rotated, and you can tell servos to go to a specific position. Most servos have a limited range of movement. All in-all, this makes them excellent for uses like grabbing arms.

Your robotics kits can power up to 12! servos through the servo board. You can see how to get your code to move your servos in the Servo board programming page.

Your robot in the simulator has 2 servos, one on each arm, perfect for grabbing!

For this task, add to the code you wrote for the previous task to make your robot grab the can with the grabbers, and drag it backwards. You'll want to take a look at the simulator programming section to see which servo is which.

Step 5 - Bump sensors 👊

Bump sensors are great for detecting if something has bumped into them. Normally, bump sensors are non-latching switches, which connect two wires when they bump into something. You could also use two bare pieces of wire if the thing you're bumping into is made of metal.

It's an excellent way of spotting when you're stuck up against a wall, or if you managed to grab something correctly.

The bump sensor in the simulator is set up very similar to how you would do it in real life, it's connected to the IO pins of the 'ruggeduino', which is a board in our robotics kit that has general purpose pins. In our physical kit you can connect up whatever you want to these pins. In the simulator, it has a bump sensor attached to digital pin 2.

As always, we have information on how to read data from the ruggeduino in our documentation: studentrobotics.org/docs/programming/sr/ruggeduinos/

For the simulator, you won't need to set the pin modes, as they are already configured for you in the virtual robot, see the simulator programming page for more info. The bump switches are in INPUT mode, and as they are only connected or disconnected, you can just '`digital_read(<pin number>)`' them.

Now, make your robot drive backwards, then stop when it hits the wall. Once it can, you can move to the next task!

Some hints with the programming:

- If you have a `R.sleep()` for more than 0.1 seconds, you're doing something wrong here.
- Look into how '`while`' loops work if you haven't heard of them before

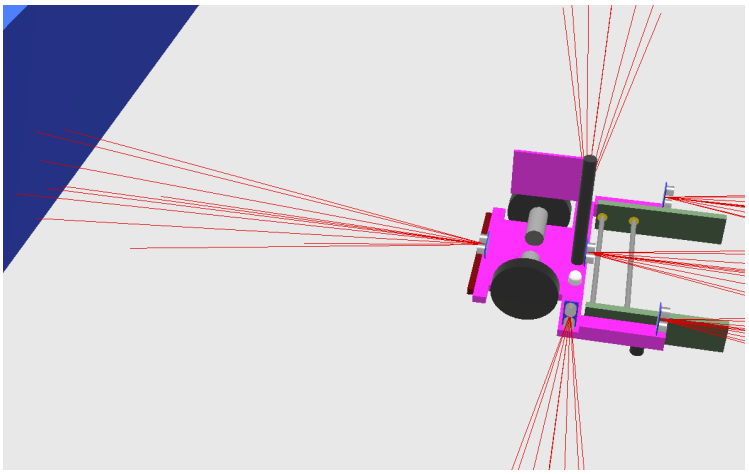
Step 6 - Light up the Sky 💡

The simulator robot has two LEDs on it, LEDs are useful to immediately see what has happened to your robot. You can hook LEDs up to your physical robot using the Arduino.

The simulator programming docs explain how to use the LEDs in the LEDs section of the simulator programming page.

Modify your program from the Microswitches task so that your robot lights its green LED when moving and its red LED when it has hit the wall.

Step 7 - Distance Sensors 👈



Distance sensors, usually ultrasonic sensors in physical robots, report the closest distance away from them in a cone perpendicular to the sensor. They can read up to 2m away, but get more noisy the further away things are. Normally they tell you the distance of the closest thing in the cone.

They can be used to detect when you're about to hit a wall. There are 6 in total, with 2 each on the front and back, so you can use them to get the angle of surfaces in front of you. As always, you can check which is which in the simulator programming section.

Our virtual robots also have the distance sensors connected up to the I/O pins of the 'ruggeduino', just like in the previous task. However these sensors give you a range of values, from 0 to 2 for distance in meters, so you will need to use `analogue_read()` instead of `digital_read()`.

You can hook up an ultrasound sensor on the physical kit too, it's a bit more involved than the other sensors because it gives you a number instead of a yes/no. Ask us on Discord and we'd be happy to explain how you'll need to do it.

To complete this task, make your robot drive forward until you are 30cm away from the wall (after bumping into it).

Step 8 - State Machines 🚦

One great way to structure your robot code is around the idea of the “states”, which the code can be in. You can then express the behaviour of your robot as being in those states and the actions it takes (or things which happen to it), which causes it to move to another state.

The idea of state machines is to keep your state in a variable, so instead of writing code like this:

```
while True:
    look_for_can()
    if (can_see_can):
        drive_to_can()
        if (can_is_near):
            pick_up_can()
            if (can_was_picked_up)
                flip_can()
                #...on to infinity
```

Yoru code can look like this:

```
while True:
    state = "look for can"

    if (state == "look for can"):
        look_for_can()
        if (can_see_can):
            state = "drive to can"

    if (state == "drive to can"):
        drive_to_can()
        if (can_is_near):
            state = "pick up can"

    if (state == "pick up can"):
        pick_up_can()
        if (can_was_picked_up):
            state = "flip can"

    if (state == "flip can"):
        flip_can()
        state = "look_for_can"
```

(Don’t be alarmed if you don’t understand this code, we’d love to explain it to you if you ask us on Discord).

The benefit of writing code like the second block is it’s much easier to swap between states instead of having to write everything in a confusing long line of ‘if’s. For example, you can go back to “look for can” if trying to pick it up fails just by adding this code:

```
        if (can_was_not_picked_up):
            state = "drive to can"
```

This is excellent programming practice, and makes your code much more flexible.

Now, try to write a state machine so that your robot uses its LEDs as a set of traffic lights. The lights should follow the following cycle:

- Red
- Red & Green
- Green
- Flashing Green
- (back to Red)

Remember to use a State variable to keep track of which one it's meant to be in!