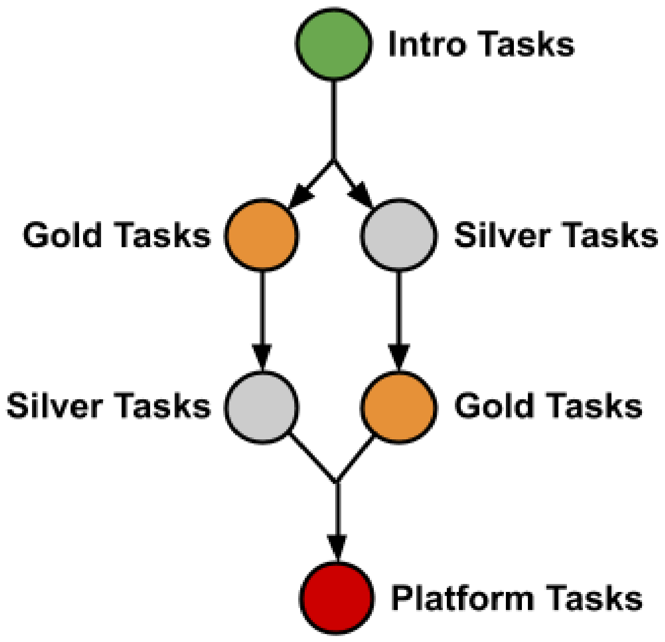
Student Robotics 2020 Microgames

Welcome to the Student Robotics 2020! In this Microgames session, you will be learning how to use our kit by completing a series of challenges.

The text in these tasks won't tell you *how* to solve the problems, you need to get used to searching through our documentation for answers to common questions, and knowing where else to look if your question still isn't answered.

You should start with the 'Intro tasks' below, then you can either move on to the Gold or Silver tasks. Once you've managed to complete both the gold and silver tasks, you can start working on the 'Platform' tasks, which are longer-form and more challenging.



Remember; the purpose of the microgames is to ensure you get to know how to use your robotics kit! Don't worry if you don't finish the tasks, all of the information you need is also available on our documentation website: studentrobotics.org/docs

🟢 Intro Tasks

Before you start building your fantastic robots, we need to get you used to the kit.

🟢 1. Battery Charging 🔋

Safety first! You **MUST** know how to safely charge the batteries we provide to you. The LiPo (Lithium Polymer) batteries we give you can contain 120 kilojoules of energy - about half a hand grenade! Before you plug a battery into the charger, make sure you've shown a volunteer what you think you should do.

The best place to start if you don't know how to do something is always the docs, at studentrobotics.org/docs. Find the instructions on how to use the battery charger in your kit. Note that there are two different models of charger and they may get hot, so be careful. Make sure you use the instructions for the one you have in your kit - check that your charger looks like the image at the top of the instructions.

You **MUST** show a volunteer before you plug a battery in. Once you have shown a volunteer and started charging a battery, move on to the next task.

☐ Volunteer has checked battery charging.

🟢 2. Assemble your kit 🏗️

You should start by assembling your kit and getting used to what's in the box. If you need some help, the first place you should always look at is the docs at studentrobotics.org/docs. At studentrobotics.org/docs/kit/assembly there's a video tutorial on how to put it all together.

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 what all of the components so you can move on to the next task.

☐ Volunteer has checked robot wiring.

3. Find the Password 🔒

Firstly, you should get some code running on your kit. The traditional program to test things are working is to print a message to the logs. To have the code run on your robot, you need to write the code in the IDE at studentrobotics.org/ide/ (your teachers should have the login details), which you then put in a USB stick and run on the robot. You can view the logs on the USB stick, or by connecting to your robot's WiFi (Check the docs studentrobotics.org/docs in the 'Wifi' section for details).

Try to get some code to run on your robot! Your challenge is to run the below code on your robot. The code will print out a secret password, your challenge is to tell a volunteer the password. Don't worry if you don't know what this code does, it's deliberately very obfuscated to make it very hard to figure out what it does without running it on the robot.

```
print "The Password is:"
print "G{1}{2}{3}{2}V{4}{0}".format(chr(ord("r")-32),
sum(''),chr(76),"B15+D"[:0:-1],int(3.9))
```

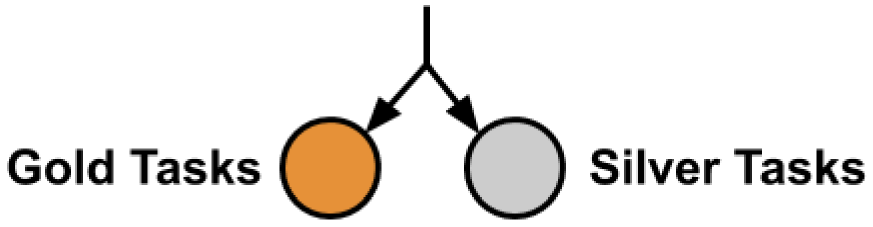
Too lazy to type it out? The code can be copied from pastebin.com/VyxG1PuJ

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

Password

☐ Volunteer has verified the password.

Once you're done, show your result to a volunteer, and they'll give you the Silver or Gold task sheet!



● Silver Tasks

● 1.Wave the robot arm 👋

For this task, you need to collect:
A Servo motor

Servos are pretty important to a successful robot design, they allow you to rotate objects to a specific angle. Servos are useful for robot arms (with a counter-weight), though we'll let your imaginations run wild for any other uses you can think of.

Connect up a servo to your robot and then get it moving by following the instructions on the docs. Show a volunteer once you're finished. Try and see what the movement limits are of the servos, they don't turn all the way!

Try to make the servo move from the center all the way to the left, then all the way to the right, in a loop.

Note: *Make sure the servo plug is the correct way round on the servo board, the darkest colour on the cable faces the bottom!*

Please return the servo afterwards as other teams will need them.

☐ Volunteer has seen the servo wiggle.

◯ 2. Post on the forum 💌

The forums are the best place to go to ask volunteers for help if things aren't working or to ask any other questions you may have. They're also a great place to chat with other competitors about their robots, and exchange ideas on strategy, design or construction. Post on the forums introducing yourselves to the other competitors, then show a volunteer.

The forums are at studentrobotics.org/forum.

If you don't have access to the forums, please speak to your team leader.

☐ Post is on the forum.

◯ 3. Use the robot simulator 🤖

Although you don't have any motors (or even a robot) yet you can still familiarise yourselves with our software using our robot simulator! Head to the docs to find the simulator. The instructions on how to use it are on the download page.

Motors can be controlled by a simple interface. A good place to start would be the docs at studentrobotics.org/docs/programming/sr/motors/.

The simulator will run most robot code you give it.

It's easy to get carried away with the simulator, so for now just see if you can get your robot to drive in a circle, there's a more advanced task later!

☐ Volunteer has seen your simulated robot drive in a circle.

● 4. Bug Hunting 🐜

Software engineers often come across many problems, be it 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 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 studentrobotics.org/docs/troubleshooting/python. If you have any other issues, ask a volunteer!

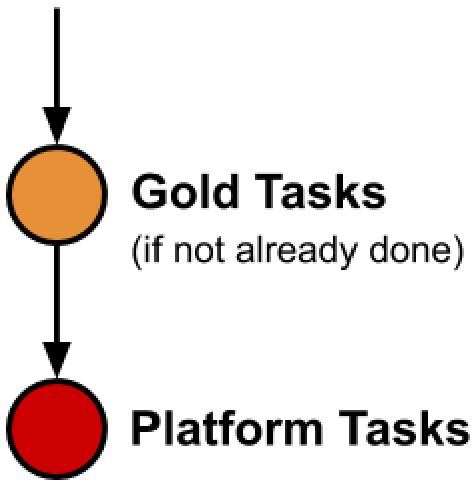
This code is available at https://pastebin.com/GGz0Eeuw if you'd like to copy paste it!

```
from sr.robot import *
Import time

R = Robot

def drive(speed, time):
    R.motors[0].m0.power = speed
    R.motors[0].m1.power = speed
    time.sleep(time)
    R.motors[0].m0.power = 0
    R.motors[0].m0.power = 0

def look_for_golden_token():
   markers = R.see()
    arena_markers = []
    for m in markers:
        if m.info.marker_types = MARKER_TOKEN_GOLD:
            arena_markers.append(m)
    return arena_markers

while True:
    markers = look_for_golden_token(R)
    # Drive forwards if a token is seen
    if len(markers) != 0:
        if markers[0].distance > 0.5:
            drive(20, 0.2) // Go forwards
```

There are 10 errors in this code, don't sweat it if you can't find them all, a few are frequently hard to spot problems we often see with students code! Once you've had a decent stab at it, ask a volunteer to verify your working.

☐ Volunteer has checked your code and explained any bugs you missed

Once you're done, show your result to a volunteer, and they'll give you the Gold task sheet, or the Platform task sheet if you've already finished Gold!



🟠 Gold Tasks

🟠 1. Lift off! 🚀

The power board has a small buzzer on it. You may be able to spot it, it's a black cylinder right next to the fan. You can use it for diagnostics (For example, the 3 beeps you hear when you turn on the kit tells us the version of the software its running is version 3) You can also use it to debug your robot code on the fly. you can program your robot to beep at a particular tone for a set length of time. In other words, your robot is a musical instrument!

Check the docs under 'Programming' and 'Power', There it will explain how to use `` `R.power.beep` ``.

Make a rocket countdown that starts by beeping 5 times with a 1 second delay between each beep, and then one long beep at the end.

For the 1 second delay, you'll need to pause your code for a given amount of time.
To do this, you can use a function in the "`time`" module. You can look up how to do this and more in the Python documentation which is available at studentrobotics.org/docs/python/.
Make sure you're always looking at the documentation for python 2, not 3!

Once that's working, show it off to a volunteer. If you ever find yourself struggling with programming, just ask a volunteer!

☐ Volunteer has seen your robot do its countdown beeps (including the long beep at the end)

🟠 2. Vision 👀

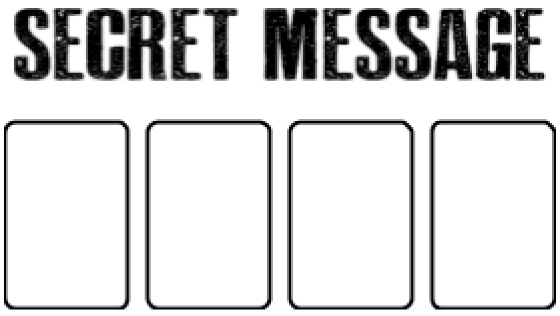
Using your robot’s camera, you’ll be able to see our visual markers around the arena. - You don’t need to write any complex computer vision code to recognise the markers though, we’ve done that for you!

Plug in the camera to your robot and write a program that looks for markers and displays information about any it can see. There’s a helpful page on the docs describing how to use our vision system.

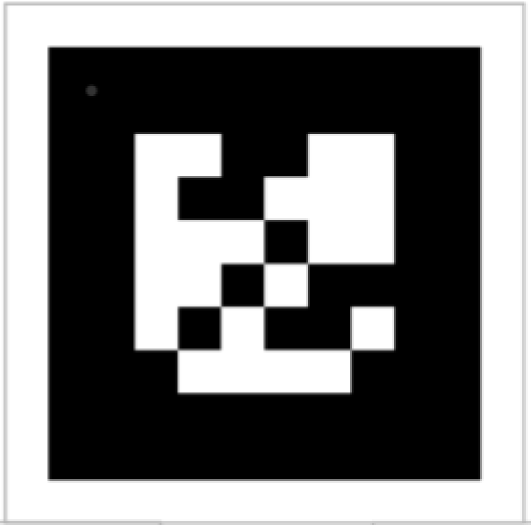
Use the pattern of 4 markers provided with this task sheet to decipher the secret 4-letter message. Each marker represents a number, convert the number to a letter, with: A = 0, B = 1, etc.. from left to right.

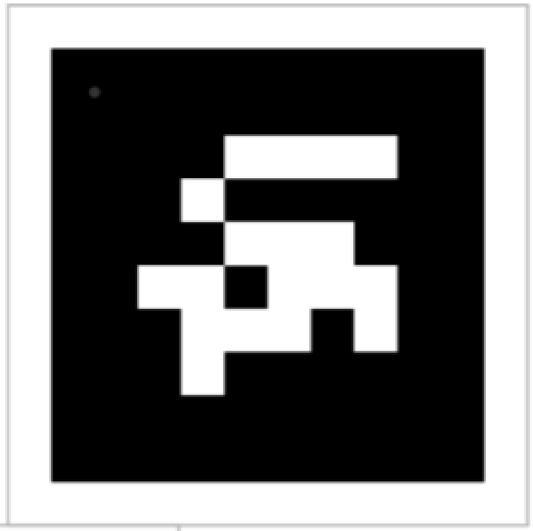
Once you’re done, write down the secret message, and get a volunteer to verify.

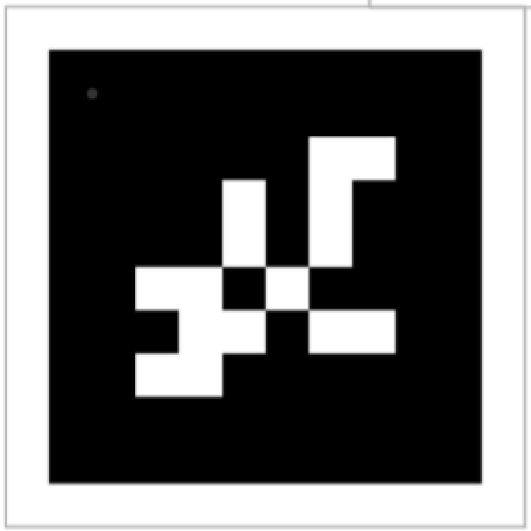
(Hint, you should use the ‘`code`’ property of `MarkerInfo`)

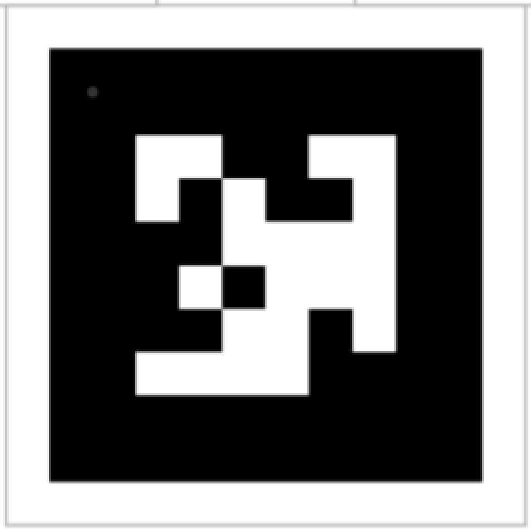


☐ Volunteer has checked your message is correct









Please note: These markers are shrunk to fit on a page, so your robot’s distance readings will be incorrect.

SECRET MESSAGE

🟠 3. Wifi Zone Control 📶

If you've plugged in the wifi dongle over USB, your robot
will automatically host a wifi hotspot, which you can connect your phone/laptop to!

You can do many things with the robot wifi, like remotely start/stop your robot, read the live logs, and set the robot starting zone.

Check the docs on how to use the robot wifi, and get your phone/laptop connected.

Your robot could start in any of the 4 starting zones in the arena corners. Your robot will need to know which corner you've started in. In the competition, we'll plug in a special USB stick that tells your robot which corner it starts in. However, for debugging purposes you'll want to be able to set this yourself.

You can do this using the webpage on the robot wifi!

Once you've connected to the robot and managed to set the robot starting zone, you'll need to write some code to use it.

You'll find the documentation on how to write code to read the robot zone in the docs, on the 'Programming'->'sr' page.

Make your robot beep based on which zone its in. Check the documentation for the power board on how to make your robot beep - in case you've forgotten since the last task!

Make the buzzer beep:

- Once for zone 0
- Twice for zone 1
- Three times for zone 2
- Four times for zone 3

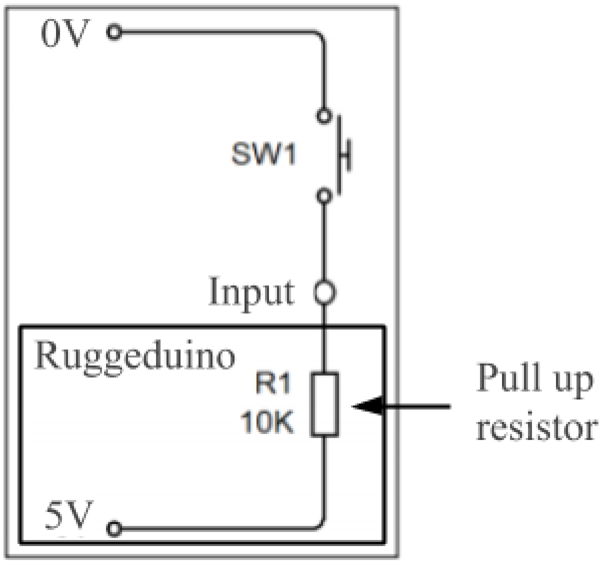
☐ Volunteer has seen your robot beep based on which starting zone it's set to.

● 4. Sensors

Even if your robot programming was perfect, you're still likely to need some input from external sensors. The simplest form of a sensor is a bump switch, which for example can be depressed when a cube bumps into your robot's grabber - very useful for checking you've grabbed a cube!

So how do you sense button presses? You need to hook them up to the ruggeduino (rugged arduino). Ruggeduinos are programmed in the C programming language, but we don't expect you to know that! All the ruggeduinos in our kit come preloaded with enough code to detect with a lot of different sensors.

On your ruggeduino, there are many pins. The relevant pins are 0V (Ground), 5V, and any of the Digital IO pins (ones numbered up to 13). You can connect wires either through the black header or through the green screw terminals. The Digital IO pins can read either 5v or 0v (it just gives you a true or false, you should use an analogue pin if you want to measure continuously between 0 and 5v).



To make a basic switch, you *could* simply connect a wire from 5v, connect a wire from one of the IO pins, then just touch them together to short the 5v to the IO pin. This would work, it would make the IO pin be 5v. However, when you disconnect it, the wire will be left 'floating', the IO pin is just a sensor, any voltage in the wire will have nowhere to go.

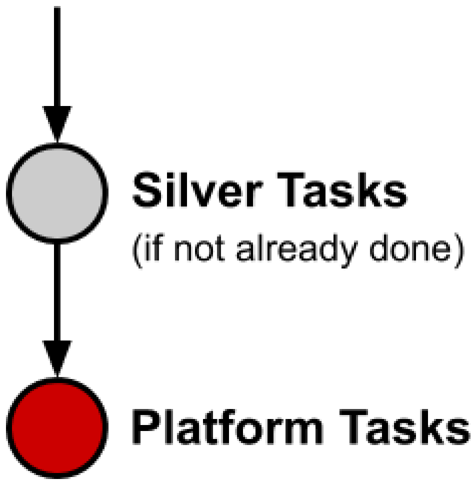
What you need is something which allows the pin to go back to 0v when it's disconnected. Thankfully, our ruggeduinos come with this already! You just set the Ruggeduino pin to the INPUT_PULLUP mode. You have to wire it backwards however (see the diagram) as the resistor means a disconnected wire will be 5V, so you should connect the other wire to 0V, so it reads 5V when disconnected and 0v when connected.

Read the docs! Under 'Programming', look at 'Ruggeduino' to find what code to use to interface with the Ruggeduino. Hint: You need to first set the `pin_mode`, then you can use `digital_read` to get the value as many times as you want.

Challenge: Write some code which makes the power board buzz when two wires (one connected to 0v and one connected to a digital IO pin) on the Ruggeduino are touched together.

☐ Volunteer has seen your robot beep when 2 wires are touched together.

Once you're done, show your result to a volunteer, and they'll give you the Silver task sheet, or the Platform task sheet if you've already finished Silver!



● Platform Tasks

You've made it to the *Platform Tasks*, These tasks are challenges to really try your skills in robot making and get you closer to the skills needed to make a successful robot.

● 1. State machines 🎰

Your robot will need to do different things at different times, for example first looking for a cube, then moving over to the cube, then picking up the cube.

There's a really neat trick to writing this kind of code which avoids spiralling into a spaghetti monster.

Let's say you have 3 functions already written (link for the lazy: pastebin.com/YJjUqfcU)

```
import random

def task_a():
      return random.randrange(2)
def task_b():
      return random.randrange(2)
def task_c():
      return random.randrange(2)
```

This code just defines 3 functions, which randomly returns either 1 or `0`.

Now let's say you need to write some code which uses these functions. Your code should start at `task_a()`, and it ceaselessly does the following:

If you ran `task_a()`:
- If you got a 1: run `task_b()`
- If you got a 0: run `task_c()`

If you ran `task_b()`:
- If you got a 1: run `task_c()`
- If you got a 0: run `task_a()`

If you ran `task_c()`:
- If you got a 1: run `task_a()`
- If you got a 0: run `task_b()`

Write some code which does the above. It might get messy, that's expected! You might stumble upon the neat trick yourself. Don't waste too much time though, Once you've taken a decent stab at starting it, call over a volunteer to see.

☐ Volunteer has seen you attempt to write some code to implement the above.

🔴 2. Advanced Simulator ⚡🤖⚡

Although it's quite basic, the simulator is a very powerful tool! Your challenge is to try to get a functional robot programmed.

Ultimately, your goal is to make a robot which can score 3 match points - That is, a robot that gets 1 cube into the scoring zone.

We'll break this down into a series of steps you should complete:

1. Make it drive forward
2. Make it see cubes
3. Make it see the closest cube
4. Make it drive to the cube
5. Make it pick up the cube (use `` `.grab()` ``, check the simulator download page on the docs)
6. Make it find home (hint: the starting zone has a wall marker with the code 0, assuming you start in zone 0)
7. Make it drive to the scoring zone and drop the cube.

If you've got your robot reliably doing this, as an extra hard bonus-challenge, you can make it so the robot knows which zone it starts in, and then run the simulator with 2 robots simultaneously (just list "robot1.py, robot2.py" when you run the simulator). Challenge your friends/other teams, and see who wins!

A word of caution however: The simulator is *not* the real world. You should always check the code works on the physical robot too!

☐ Volunteer has seen your amazing robot simulation score 3 match points (gets a cube to the scoring zone).

● 3. Listen for the competition (ADVANCED) 👂

***Note:** collect an ultrasound sensor and four hookup wires from the volunteer desk*

Sometimes you may find yourself limited by the capabilities of our kit, so we try to make it easy for you to add your own extensions to it. In this task, you'll add the ability to measure the distance of an object from an ultrasonic proximity sensor, using the Ruggeduino. This will be useful for finding the distance to cubes or other objects in the final competition.

You'll first need to download and install the Arduino IDE. You can get it from arduino.cc/en/Main/Software. If you have trouble installing it, check the Arduino website. If you're still stuck, ask a volunteer!

Next, try adding a custom command to the Ruggeduino that does something simple, like blink an LED. The docs page studentrobotics.org/docs/programming/sr/ruggeduinos/custom_firmware may be very helpful for this.

The ultrasonic sensors you'll be using have four pins on them, labelled Vcc, Trig, Echo and Gnd. Their connections to the Ruggeduino should be made as follows:

- Vcc (sensor) to 5V (Ruggeduino)
- Trig (sensor) to 2 (Ruggeduino)
- Echo (sensor) to 3 (Ruggeduino)
- Gnd (sensor) to Gnd (Ruggeduino)

It's operated by sending a short pulse to the sensor's Trig pin. The sensor then generates a pulse on the Echo pin in response; the duration of this pulse indicates how far the away an object in front of the sensor is. Specifically, the duration of the pulse you send should be 10 microseconds, and the duration of the pulse you receive in response is equal to **round-trip distance** between the object and sensor divided by the speed of sound (343 metres per second).

Some Arduino functions that you might find useful for implementing this are:

- `pinMode`
- `digitalWrite`
- `pulseIn`
- `delayMicroseconds`

Their names are fairly self-explanatory, but more detailed information can be found in the Arduino documentation: https://www.arduino.cc/reference/en/