

# Student Robotics 2023 Microgames

Welcome to the Student Robotics 2023 microgames! You will be learning how to use our kit by completing a series of challenges.

## Task 0 - Join our Discord

Our Discord server is *the best* way to chat with us and other competitors. Our friendly volunteers are able to help you solve any problems you 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, join the discord forum with the invite link below, be sure to use 1 account per person.

1. Click or scan the join link here:



When you're in, you'll be taken to the [#welcome-and-rules](#) channel. You will not see any other channels by default, in order to get in, you need to enter your team's password:

2. Click on the '#welcome-<username>' channel (it should have a red badge on it), you need to read and accept our rules by clicking on the button at the bottom before you can type anything in that chat.
3. **Enter your team's password** into the '#welcome-<username>' channel. If the message is correct you'll immediately get access. Your supervisor should have the password for your team.

Task Summary:

Join our discord with the link above, then **send a message in #say-hello**.


Done? Now you can start our challenges, you need to do either the 'Kit Section' of the microgames if you have your kit, alternatively, we've prepared a simulator version in the 'Simulator' section.

# Kit Section

This bit of the microgames teaches you the basics on how to use your Student Robotics kit. Don't have your kit yet? Feel free to skip ahead to the [Simulator section](#) to get acquainted with our API and Simulator.

## Task 1 - Battery Charging

We take battery safety very seriously

The two Lithium Polymer (LiPo) batteries in your kit together store the same energy as a **hand grenade** , so it's very important to treat them with care. Having said that, if treated properly these batteries are very safe. Please ensure you:

- Never pierce your batteries.
- Don't drop your batteries.
- Don't leave your batteries to tumble loosely in your robot when running it.

If you do treat these badly, batteries can enter 'thermal runaway', where the heat of a short-circuit can break down the insulation between cells and cause further shorts. When a battery enters thermal runaway, it's very hard to stop. So, please do treat them with care.

If you don't know how to do something, the best place to start is our documentation, at [studentrobotics.org/docs](http://studentrobotics.org/docs). In the docs, we have detailed instructions on how to use the battery chargers in your kit.

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

Task Summary:

Connect a battery and start charging it in the battery bag. Read the instructions in our docs, **the obvious way to charge is not the correct way**. Please get someone to check things are plugged in correctly before you start charging!

## Task 2 - Assemble your kit 🛠️

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

Our kits have been updated this year, so even if you competed last year, it's good to pay attention!

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](https://studentrobotics.org/docs). It's very important to follow the instructions carefully.

Before you can assemble your kit, you will need to make your power cables. You'll need:

- The small flathead screwdriver in your kit
- Some black and red wire that should be in your kit
- Some wire cutters/strippers

Strip about 0.5cm off the end of short lengths of red and black cables, then screw them into the green screw connector (which we call Camcons), the ones that are the right size to fit into your power board L0/L1/H0/H1..etc sockets. Make sure you match red to '+' and black to '-'.

Once you've made these, you can start plugging it all in, at our [Assembly page in the docs](#), there's a video tutorial on how to put it all together. If you have any more questions, don't hesitate to ask in person or in Discord.

Note: Your brain board must be plugged into the 'L2' socket on the power board for it to receive power.

Note: Your kit also won't turn on unless you have a jumper cable 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. The thickness of the cable does not matter.

**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.**

Take your time, once you're happy that you've assembled your kit, it powers on, and you know which board is which, you can move on to the next task.

Task Summary:  
Assemble your kit!

## Task 3 - Hello Kit 🙌🧰

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

The purpose of this task is to learn how to program your robot!

In order to put code on a robot, all you need is:

- A computer with a USB-A port (see picture)
- The USB stick included in your kit.



You can follow the instructions on how to program the robot in the ['Getting Code on the robot'](#) section of the docs.

Now, in order to put code on your robot, you first need to write it! Our robots are programmed in python, we recommend you use a code editor to write your code, 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", write this line into a robot.py file in the root of your USB stick, then plug it into the robot!

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

Task: Once you're printing "Hello, World!", you can try and find the secret password, add the lines below and see what it prints. This code is deliberately complex and unreadable, so you only get the correct answer if your robot is set-up correctly and only one motor board is plugged in

```
from sr.robot3 import Robot as R
print(f"{str(len(dir(int))-2)}kn{R(auto_start=True).motor_board.serial_number[0]}")
```

*The code is also available at [pastebin.com/vsCX4YXC](https://pastebin.com/vsCX4YXC) for you to copy/paste on a device.*

You can check in your team channel on Discord on what the secret code is.

Task Summary:

Find the secret code!

## Task 4 - Change LED colours! 🚧

Once you've got your code ready, let's see if you can write some code that makes your robot do something! We'll start off easy, blinking an LED (Light emitting diode)!

New to SR2023, your brain board is now a Raspberry Pi, with the 'KCH' hat, which has controllable LEDs, you can control these very easily in your code. We've found LEDs are really useful to quickly figure out what's happening in your code, so please don't forget to use them when you're testing your real robot code!

Firstly, you need the two lines of code to firstly import what you need, and secondly to set up your robot:

```
from sr.robot3 import *  
R = Robot()
```

*Note: Your code editor may complain about 'sr.robot3' not existing. You can ignore this error, it's just that you don't have (and don't need) the robot code installed on your PC. If it bugs you, you can fix this in some editors, check the [Code Editors](#) page in our docs.*

After the above lines, you can turn each of the Red, Green, and Blue channels of each LED on or off, like so:

```
# Make the 'A' LED Red  
R.kch.leds[UserLED.A].r = True
```

Please check the [LED page in our docs](#) for all the things you can do!

Once you've got the code running, make all of the LEDs switch between **Red**, **Green**, then **Blue** every second.

Hints:

- To wait for 1 second you'll need to use the [time.sleep\(\)](#) method (which requires you add "import time" to the top of your file).
- You can use a ['while True'](#) loop to make your code run forever

Task Summary:

Make LEDs A, B, and C switch between Red, Green and Blue every 1.0 seconds.

## Task 5 - Wave that Servo!

Note: for this task you will need a Servo, which is not included in your kit. If you're at kickstart, you can ask a volunteer to let you borrow one.

Now you've got your code running, the next step is to get your robot to move something! Your robot has a servo board and two motor boards just for this purpose, this task focuses on the **servo board**.

A servo is a motor that is aware of its position (note: there are also 'continuous' servos which track their speed instead), this is normally done using a small circuit board with a [potentiometer](#) on the output. Normally, servos cannot move more than 180 degrees, so they're perfect for a grabber or arm.

1. Plug the 3-pin connector of your servo into the first of the 12 outputs of the servo board. Hint: the darker coloured wire faces the bottom of the board.
2. Check the [Servo programming page](#) on the docs to add code to your robot to move the servo, hint: the code to move the first servo looks like this:

```
R.servo_board.servos[0].position = 0.2
```

3. Make the servo wave back and forth every 0.5 seconds. Our servo code does not wait for the servo to move to its location, so you will need to use the [time.sleep\(\)](#) function if the code afterwards relies on the servo being in its final position.

Once your servo is happily waving away, you can move on to the next task!

Task Summary:

Make a servo wave back and forth every 0.5 seconds.

## Task 6 - Wire up a switch!

Note: for this task you will need:

- Some hookup wires
- A switch
- Breadboard to wire it all up

Most of this is not included in your kit, if you're at kickstart you can ask a volunteer to let you borrow these things. **Please note we have very limited supply, please return this as soon as you're finished with it.**

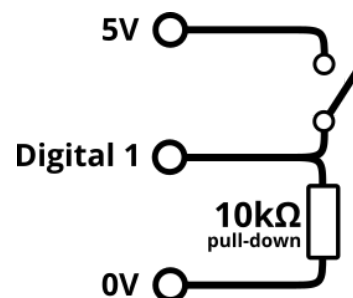
In your kit is what we call a 'Ruggeduino', it's a more rugged version of an arduino, if you've heard of it. The Ruggeduino in our kit is used as a general purpose input/output device. You can wire up many different sensors to your ruggeduino; bump switches, ultrasound sensors to measure distance.

With this task we're going to wire up a switch to the ruggeduino, and make the robot react to it. On your ruggeduino, you have a bunch of pins, here are the most important:

- **'5V'** gives out a constant 5 volts.
- **GND** is Ground, or 0V.
- **'Digital' pins** (labelled 2-13) read whether the voltage is 5V, or 0V, it just gives you a true or a false, they can also be configured to set or 'write' their voltage to 3.3v or 0. This is useful if you want to control something, like an LED.
- **'A' pins** (A0-5) can read the voltage, they give you the voltage between 0 and 5 as a non-integer.

To make a basic switch, you could simply connect a wire from 5v, connect a wire from one of the digital 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', if you were to hold your Digital pin wire next to something electrically noisy (like a motor) it's possible the arduino will give you a false positive.

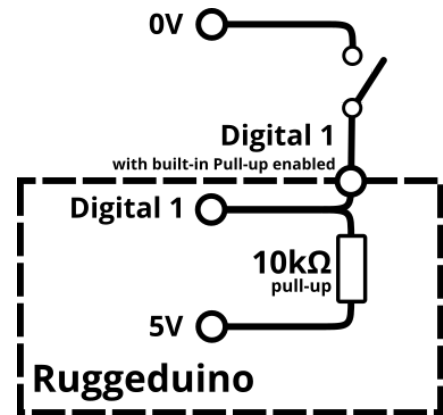
So what do we do? We also connect the pin to GND to pull down any voltage from electrical noise, and to stop the current flowing directly from 5V to GND, we put a big 10kΩ resistor on it. This improves the quality of the signal you get from the switch.





Thankfully, this is made much easier with our robotics kit. Our Ruggeduino supports built-in pull up and pull-down resistors. Also, it's a bit more reliable to have a wire be 5V when the switch is open, and 0V when the switch is closed, so we'll swap the 0V and 5V around. That gives us the diagram to the right.

So all you need to do is wire one side of the switch to 0V, and the other to a Digital pin, and you should have a working switch if you enable the built-in pull-up resistor.



Once you've got it wired up, you can program your robot to read it. By now you should be a pro at reading [our docs](#) and figuring out what code to write, so we won't give you much guidance here. Just remember that you need to *enable* the pull-up resistor by setting the pin mode to 'INPUT\_PULLUP'.

Task Summary:

Make LED A turn **Green** when you press the switch

## Task 7 - 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 the code to the right. Point out any errors, and then fix them. Copy the code (also found on <https://pastebin.com/R6cBF45m>) into your code editor, then find the fixes, and then run your code on your kit!

You can also run it on the simulator, which can be found on the docs, but you will need to remove the '#' in line 10 and add a '#' before line 8 in your code.

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 in person or in Discord!

Hint: there are 7 errors in total.

Task Summary:

Find the 7 bugs in the above code

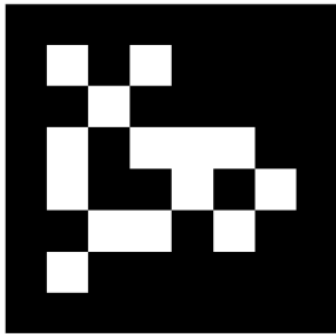
```
1 from sr.robot3 import *
2 import time
3
4 R = Robot()
5
6 def sleep(duration):
7     # if using a kit, use this:
8     time.sleep(duration)
9     # if using the simulator, use this
10    # R.sleep(duration)
11
12 def spin(duration, speed):
13    # make robot spin
14    R.motor_board.motors[0].power = speed
15    R.motor_board.motors[0].power = -speed
16    sleep(duration)
17    R.motor_board.motors[0].power = 0
18    R.motor_board.motors[1].power = "0"
19
20 def look_for_any_marker()
21    marker_ids = R.camera.see_ids()
22    if len(marker_id) > 0:
23        print "Found a marker!"
24        return marker_ids[0]
25
26 marker = None
27 while marker is None:
28    print("Spinning to find a marker")
29    duration = 0.1
30    speed = 0.5
31    spin(speed, duration)
32    marker = look_for_any_marker()
```

## Task 8 - Look for things 🧐

Your robot has a camera! With this, your robot can spot markers from far away. Our robot API does the complex vision work, and outputs the relative position of the markers in addition to both angle of the face relative to the robot, and angle of the marker relative to the robot. You can read [our docs](#) to find out more about vision.

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

↑ **Secret Code - This way up** ↑

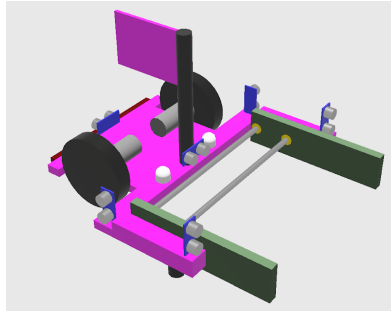


[Click here for a bigger version.](#)

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



This section of the microgames uses the simulator, rather than the physical kit. We've prepared a basic robot simulator to help you learn to program your robot. We've designed the simulator to be as similar to using a real robot as we can. This year you'll be scoring points with your simulator robots too, so it's extra important this year!

## Task 1 - Install the Simulator

First, you need to get the simulator working! Our Simulator supports Mac, Windows, and Linux.

Follow the instructions on [the Simulator page in our docs](#). Our documentation should be your go-to place to find information about our robots, so you should learn how to read and navigate it well!

You will need to download both [Webots](#) and [our simulator](#) files, and **make sure you have the latest version of Python installed** (at least python 3.8) installed on your machine.

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. In the 'Worlds' folder, double click on the arena .wbt file to run the simulation. It should launch webots (if not, ask for help!), then start running a simulation of our sample robot. If you get an error, ask for help, usually it's an issue with your python installation.

After successfully running the simulation, 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).

Now, in order to run code on your robot, you first need to write it! Our robots are programmed in python, we recommend you use a code editor to write your code, so check out our [Code Editors](#) page in the docs to see what you could use.

## Task 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:", "sinLeikmoiTNlwapecteS"[::-2])
```

This code is also available at [pastebin.com/1NScE5fe](https://pastebin.com/1NScE5fe) if you have this printed out.

(Don't worry if you don't understand it, we made it to be deliberately hard to understand, so it's tricky to figure out what it does before you run it, kudos to those who do understand it however.)

Then, after saving, you can run the code by double-clicking on the arena .wbt file (in the 'worlds' folder).

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 3.8. You can install a later version at [www.python.org/downloads/](https://www.python.org/downloads/).*

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

## Task 3 - 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 has 1 motor board with 2 motors plugged in, in your physical kit we give you two motor boards, meaning you can power up to 4 motors for your actual robot!

Now, make the robot drive forwards for 4 seconds, then drive backwards for 2 seconds.

**Important:** In our simulator, you should avoid using the normal python way to wait (`time.sleep()`), as it would change your robot's performance if you're on a laggy machine. Instead you should use `Robot.sleep()`, like this:

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

Use this to make the code wait for a given amount of time. Remember, motors only ever change speed if you tell them to, so if you set the speed to 0.5, then sleep for 1 second, then set the speed to 0, the robot will have moved forward at 0.5 speed for 1 second..

## Task 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 token 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.

## Task 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/](http://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

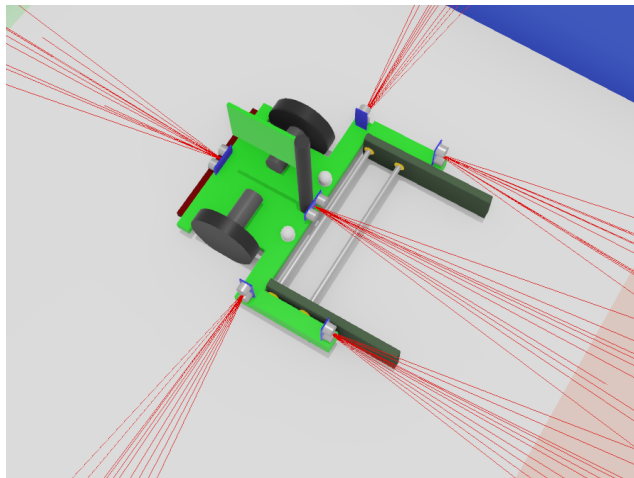
## Task 6 - Light up the Sky 💡

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.

## Task 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 metres, 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).

## Task 8 - Find & pick up a cube 🗉👉📦

Let's put together what you've learned so far into a challenge, your goal is to make a robot which can score 3 match points - That is, a robot that gets 1 bronze cube from the opponents scoring zone into your scoring zone.

1. 🔄 use your knowledge of motors to make your robot turn around to roughly face the cubes.
2. 👁️ Use vision to spot the cubes, use [our vision documentation](#) to see how to identify markers. Do make note of [what the rules say](#) about Marker IDs, or you'll end up spotting a wall instead of a cube.
3. Find the closest token to the camera.  
*Hint: you can do this using `R.camera.see()`, which returns a list of Marker objects, then use `m.distance` (the distance property on the marker object), which reports the distance of the marker from the camera.*
4. 👉 Use the motors to drive towards the token.
5. 📦 Use the 'Front' distance sensor to find when the token is servos to close the grabber around the token.

If you've got your robot reliably doing this, you've finished this task! Be sure to boast about it in discord, we'd love to see it!

## Task 9 - 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_token()
    if (can_see_token):
        drive_to_token()
        if (token_is_near):
            grab_token()
```

```
if (token_was_grabbed)
    look_for_token()
    #...on to infinity
```

Your code can look like this:

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

    if (state == "look for token"):
        look_for_token()
        if (can_see_token):
            state = "drive to token"

    if (state == "drive to token"):
        drive_to_token()
        if (token_is_near):
            state = "grab token"

    if (state == "grab token"):
        pick_up_token()
        if (token_was_picked_up):
            state = "flip token"

    if (state == "flip token"):
        flip_token()
        state = "look_for_token"
```

(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 'ifs'. For example, you can go back to "look for can" if trying to pick it up fails just by adding this code:

```
if (token_was_not_grabbed):
    state = "drive to token"
```

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!