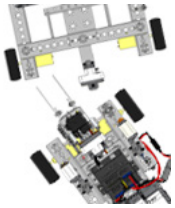
ROBOTC Natural Language - TETRIX Reference:

Setup Functions:

Robot Type
Choose which robot you want to write a program for. Note that not including this command defaults to "`robotType(none);`" Also please note that this command should be the first thing in your "`task main()`".



Command:

```
robotType(type);
```

Parameters: `type`

Valid Robot Types for `type`:
`none` - this will not set up any motors and sensors for you (this is the default.)
`mantis` - sets the motors and sensors to match a default MANTIS.
`ranger` - sets the motors and sensors to match a default RANGER.

Usage without Parameters:

```
robotType();
```

This snippet of code will set the robot type to **`none`** by default, skipping the setup process. You must manually set the motors and sensors in the 'Motors and Sensors Setup' menu.

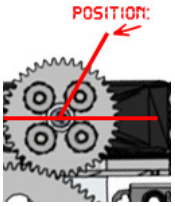
Usage with Parameters:

```
robotType(mantis);
```

This snippet of code will set the robot type to **`mantis`**. This will automatically set up the motor and sensor ports to match those of a default MANTIS.

Movement Functions:

Set Servo
Set a servo to a desired position.



Command:

```
setServo(servo, position);
```

Parameters: `servo`, `position`

Acceptable Motors for `servo`:
SERVO ports 1 through 24 (and your names for them given in Motors and Sensors Setup.)

Valid Range Values for `position`:
0 to 255.

Usage without Parameters:

```
setServo();
```

This snippet of code will set the servo on servo-port 1 to position 0 (center). The default servo-port is **`srvo_S1_C1_1`** and the default position is **`0`** for **`setServo()`**.

Usage with Parameters:

```
setServo(srvo_S1_C1_6, 37);
```

This snippet of code will set the servo on servo-port 6 to position 37.

ROBOTC Natural Language - TETRIX Reference:

Start Motor
Set a motor to a speed.

Command:

```
startMotor(motor, speed);
```

Parameters: `motor`, `speed`

Acceptable Motors for `motor`:
MOTOR ports A through K (and your names for them given in Motors and Sensors Setup.)

Valid Range Values for `speed`:
(reverse) -100 to 100 (forward) where 0 is stop.

Usage without Parameters:

```
startMotor();
wait();
stopMotor();
```

This snippet of code will run the motor in motor-port A at speed 75 for 1.0 seconds and then stop it. The default motor-port is **`motorA`** and the default speed is **`75`** for **`startMotor()`**.

Usage with Parameters:

```
startMotor(motorF, -25);
wait(0.5);
stopMotor(motorF);
```

This snippet of code will run the motor in motor-port F at speed -25 for 0.5 seconds and then stop it.

Stop Motor
Stops a motor.

Command:

```
stopMotor(motor);
```

Parameters: `motor`

Acceptable Motors for `motor`:
MOTOR ports A through K (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
startMotor();
wait();
stopMotor();
```

This snippet of code will run the motor in motor-port A at speed 75 for 1.0 seconds and then stop it. The default motor-port is **`motorA`** for **`stopMotor()`**.

Usage with Parameters:

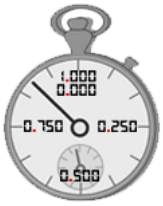
```
startMotor(motorF, -25);
wait(0.5);
stopMotor(motorF);
```

This snippet of code will run the motor in motor-port F at speed -25 for 0.5 seconds and then stop it.

ROBOTC Natural Language - TETRIX Reference:

Wait Functions:

Wait
Wait an amount of time measured in seconds. The robot continues to do what it was doing during this time.



Command:

```
wait(time);
```

Parameters: `time`

Valid Range Values for `time`**:**
0.0 to 3600.0 and up.

Usage without Parameters:

```
forward();
wait();
stop();
```

This snippet of code will run the robot forward for 1.0 seconds and then stop. The default time is `1.0` (seconds) for `wait()`.

Usage with Parameters:

```
forward(50);
wait(2.73);
stop();
```

This snippet of code will run the robot forward at half speed for 2.73 seconds and then stop.

Wait in Milliseconds
Wait an amount of time in milliseconds. The robot continues to do what it was doing during this time.

Command:

```
waitInMilliseconds(time);
```

Parameters: `time`

Valid Range Values for `time`**:**
0 to 3600000 and up.

Usage without Parameters:

```
forward();
waitInMilliseconds();
stop();
```

This snippet of code will run the robot forward for 1000 milliseconds (1.0 seconds) and then stop. The default time is `1000` (milliseconds) for `waitInMilliseconds()`.

Usage with Parameters:

```
forward(50);
waitInMilliseconds(2730);
stop();
```

This snippet of code will run the robot forward at half speed for 2730 milliseconds (2.73 seconds) and then stop.

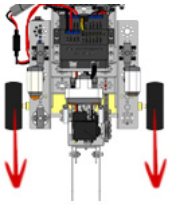
ROBOTC Natural Language - TETRIX Reference:

Robot Movement Functions:

*Note that for desirable results with the following set of functions, you must use the "**`robotType();`**" Setup Function with type **`mantis`** or **`ranger`** in the beginning of your "**`task main()`**".*

Forward

Both wheels rotate forward at the same speed, causing the robot to move forward.



Command:

```
forward(speed);
```

Parameters: `speed`

Valid Range Values for `speed`:
0 to 100 (however **`forward()`** will always move your robot forward.)

Usage without Parameters:

```
forward();
wait();
stop();
```

This snippet of code will run the robot forward for 1.0 seconds and then stop. The default speed is `75` for **`forward()`**.

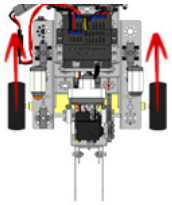
Usage with Parameters:

```
forward(50);
wait(2.0);
stop();
```

This snippet of code will run the robot forward at half speed for 2.0 seconds and then stop.

Backward

Both wheels rotate backward at the same speed, causing the robot to move backward.



Command:

```
backward(speed);
```

Parameters: `speed`

Valid Range Values for `speed`:
-100 to 0 (however **`backward()`** will always move your robot backward.)

Usage without Parameters:

```
backward();
wait();
stop();
```

This snippet of code will run the robot backward for 1.0 seconds and then stop. The default speed is `-75` for **`backward()`**.

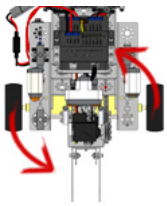
Usage with Parameters:

```
backward(-50);
wait(2.0);
stop();
```

This snippet of code will run the robot backward at half speed for 2.0 seconds and then stop.

ROBOTC Natural Language - TETRIX Reference:

Point Turn
Both wheels rotate at the same speed but in opposite directions, causing the robot to turn in place.



Command:

```
pointTurn(direction, speed);
```

Parameters: `direction`, `speed`

Valid Directions for `direction`:
`left` and `right`.

Valid Range Values for `speed`:
-100 to 100.

Usage without Parameters:

```
pointTurn();
wait();
stop();
```

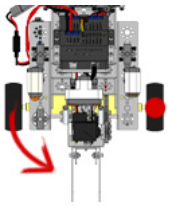
This snippet of code will make the robot turn right in place at speed 75 for 1.0 seconds and then stop. The default direction and speed are `right` and `75` for `pointTurn()`.

Usage with Parameters:

```
pointTurn(left, 50);
wait(0.4);
stop();
```

This snippet of code will make the robot turn left in place at half speed for 0.4 seconds.

Swing Turn
One wheel rotates while the other does not move, causing the robot to make a wide turn around the stopped wheel.



Command:

```
swingTurn(direction, speed);
```

Parameters: `direction`, `speed`

Valid Directions for `direction`:
`left` and `right`.

Valid Range Values for `speed`:
-100 to 100.

Usage without Parameters:

```
swingTurn();
wait();
stop();
```

This snippet of code will make the robot make a wide right turn at speed 75 for 1.0 seconds and then stop. The default direction and speed are `right` and `75` for `swingTurn()`.

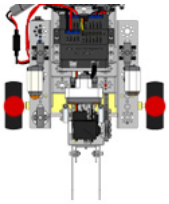
Usage with Parameters:

```
swingTurn(left, 50);
wait(0.75);
stop();
```

This snippet of code will make the robot make a wide left turn at half speed for 0.75 seconds.

ROBOTC Natural Language - TETRIX Reference:

Stop
Both wheels do not move, causing the robot to stop.



Command:

```
stop();
```

Parameters: N/A

Usage without Parameters:

```
forward();
wait();
stop();
```

This snippet of code will run the robot forward for 1.0 seconds and then stop. (Note that there are no parameters for `stop()`.

Usage with Parameters:

```
forward(50);
wait(2.0);
stop();
```

This snippet of code will run the robot forward at half speed for 2.0 seconds and then stop.

Line Track for Time
The robot will track a dark line on a light surface for a specified time in seconds.

Command:

```
lineTrackForTime(time, threshold, sensorPort);
```

Parameters: `time`, `threshold`, `sensorPort`

Valid Range Values for `time`:
0 to 3600.0 and up.

Valid Range Values for `threshold`:
(dark) 0 to 100 (light).

Acceptable Sensors for `sensorPort`:
SENSOR ports 1 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
lineTrackForTime();
stop();
```

This snippet of code will make the robot follow a dark line on a light surface for `5.0` seconds using a threshold of `45` and a light sensor in sensor-port `S3` and then stop. These values and sensors are the defaults for `lineTrackForTime()`.

Usage with Parameters:

```
lineTrackForTime(7.5, 75, S2);
stop();
```

This snippet of code will make the robot follow a dark line on a light surface for 7.5 seconds, using a threshold of 75 and a light sensor in sensor-ports 2 and then stop.

ROBOTC Natural Language - TETRIX Reference:

Line Track for Rotations
The robot will track a dark line on a light surface for a specified distance in encoder rotations.

Command:

```
lineTrackForRotations(rotations, threshold, sensorPort);
```

Parameters: `rotations`, `threshold`, `sensorPort`

Valid Range Values for `rotations`:
0 to 65000.0 and up.

Valid Range Values for `threshold`:
(dark) 0 to 100 (light).

Acceptable Sensors for `sensorPort`:
SENSOR ports 1 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
lineTrackForRotations();
stop();
```

This snippet of code will make the robot follow a dark line on a white surface for `3.0` rotations using a threshold of `45` and a light sensor in sensor-ports `S3` and then stop. These values and sensors are the defaults for `lineTrackForRotations()`.

Usage with Parameters:

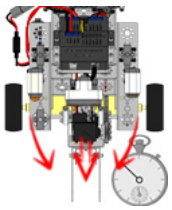
```
lineTrackForRotations(4.75, 75, S2);
stop();
```

This snippet of code will make the robot follow a dark line on a white surface for 4.75 rotations, using a threshold of 75 and a light sensors in sensor-port S2 and then stop.

ROBOTC Natural Language - TETRIX Reference:

Move Straight for Time

The robot will use encoders to maintain a straight course for a specified length of time in seconds.



Command:

```
moveStraightForTime(time, rightMotorEncoder, leftMotorEncoder);
```

Parameters: `time`, `rightMotorEncoder`, `leftMotorEncoder`

Valid Range Values for `time`:
0 to 3600.0 and up.

Acceptable Sensors for `rightMotorEncoder`, `leftMotorEncoder`:
MOTOR ports A through K (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
moveStraightForTime();
stop();
```

This snippet of code will make the robot move forward, maintaining a straight heading for `5.0` seconds using the built-in encoders on motor-ports `motorD` and `motorE`, and then stop. These values and sensors are the defaults for `moveStraightForTime()`.

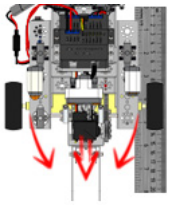
Usage with Parameters:

```
moveStraightForTime(7.5, motorF, motorB);
stop();
```

This snippet of code will make the robot move forward, maintaining a straight heading for 7.5 seconds using the built-in encoders on motor-ports F and B, and then stop.

Move Straight for Rotations

The robot will use encoders to maintain a straight course for a specified distance in rotations.



Command:

```
moveStraightForRotations(time, rightMotorEncoder, leftMotorEncoder);
```

Parameters: `rotations`, `rightMotorEncoder`, `leftMotorEncoder`

Valid Range Values for `rotaions`:
0 to 65000.0 and up.

Acceptable Sensors for `rightEncoder`, `leftEncoder`:
MOTOR ports A through K (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
moveStraightForRotations();
stop();
```

This snippet of code will make the robot move forward, maintaining a straight heading for `1.0` rotations using the built-in encoders on motor-ports `motorD` and `motorE`, and then stop. These values and sensors are the defaults for `moveStraightForRotations()`.

Usage with Parameters:

```
moveStraightForRotations(4.75, motorF, motorB);
stop();
```

This snippet of code will make the robot move forward, maintaining a straight heading for 4.75 rotations using the built-in encoders on motor-ports F and B, and then stop.

ROBOTC Natural Language - TETRIX Reference:

Tank Control
The robot will be remote controlled in such a way that the right motor is mapped to the right joystick and the left motor is mapped to the left joystick.

Command:

```
tankControl(rightJoystick, leftJoystick, threshold);
```

Parameters: `rightJoystick`, `leftJoystick`, `threshold`

Valid Channels for `rightJoystick, leftJoystick`:
Any of the `"joystick."` channels will work, however `joystick.joy1_y2` and `joystick.joy1_y1` make the most sense for this application.

Usage without Parameters:

```
while(true)
{
  tankControl();
}
```

This snippet of code will remote control the robot using "tank control" with a "dead" threshold of 10. The default right and left joysticks and threhold are `joystick.joy1_y2`, `joystick.joy1_y1`, and `10` for `tankControl()`.

Usage with Parameters:

```
while(true)
{
  tankControl(joystick.joy1_y1, joystick.joy1_y2, 5);
}
```

This snippet of code will remote control the robot using "tank control" with channel joy1_y1 as the right joystick and channel joy1_y2 as the left joystick with a threshold of 5.

Arcade Control
The robot will be remote controlled in such a way that the movement of the robot is mapped to a single joystick, much like a retro arcade game.

Command:

```
arcadeControl(verticalJoystick, horizontalJoystick, threshold);
```

Parameters: `verticalJoystick`, `horizontalJoystick`, `threshold`

Valid Channels for `verticalJoystick, horizontalJoystick`:
Any of the `"joystick."` channels will work, however `joystick.joy1_y2` and `joystick.joy1_x2` make the most sense for this application.

Usage without Parameters:

```
while(true)
{
  arcadeControl();
}
```

This snippet of code will remote control the robot using "arcade control" with a "dead" threshold of 10. The default vertical and horizontal joysticks and threhold are `joystick.joy1_y2`, `joystick.joy1_x2`, and `10` for `tankControl()`.

Usage with Parameters:

```
while(true)
{
  arcadeControl(joystick.joy1_y1, joystick.joy1_y2, 5);
}
```

This snippet of code will remote control the robot using "arcade control" with channel joy1_y1 as the vertical joystick and channel joy1_x1 as the horizontal joystick with a threshold of 5. (Uses the left joystick where default uses the right.)

ROBOTC Natural Language - TETRIX Reference:

Until Functions:

Until Touch
The robot continues what it was doing until the touch sensor is pressed in.

Command:

```
untilTouch(sensorPort);
```

Parameters: `sensorPort`

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilTouch();
stop();
```

This snippet of code will run the robot forward until the touch sensor in sensor-port 1 is pressed, and then stop. The default sensor port is `S2` for `untilTouch()`.

Usage with Parameters:

```
forward(50);
untilTouch(S3);
stop();
```

This snippet of code will run the robot forward at half speed until the touch sensor in sensor-port 3 is pressed, and then stop.

Until Release
The robot continues what it was doing until the touch sensor is released out.

Command:

```
untilRelease(sensorPort);
```

Parameters: `sensorPort`

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilRelease();
stop();
```

This snippet of code will run the robot forward until the touch sensor in sensor-port 1 is released, and then stop. The default sensor port is `S2` for `untilRelease()`.

Usage with Parameters:

```
forward(50);
untilRelease(S3);
stop();
```

This snippet of code will run the robot forward at half speed until the touch sensor in sensor-port 3 is released, and then stop.

ROBOTC Natural Language - TETRIX Reference:

Until Bump
The robot continues what it was doing until the touch sensor is pressed in and then released out.
(A delay time in milliseconds can be specified as well.)

Command:

```
untilBump(sensorPort, delayTimeMS);
```

Parameters: `sensorPort, delayTimeMS`

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Valid Range Values for `delayTimeMS`:
0 to 3600000 and up.

Usage without Parameters:

```
forward();
untilBump();
stop();
```

This snippet of code will run the robot forward until the touch sensor in sensor-port 1 is pressed in and then released out, and then stop. The default sensor port and delay time are `S2` and `10` for `untilBump()`.

Usage with Parameters:

```
forward(50);
untilBump(S3, 100);
stop();
```

This snippet of code will run the robot forward at half speed until the touch sensor in sensor-port 3 is pressed in and then released out (with a delay of 100ms), and then stop.

Until Button Press
The robot continues what it was doing until a specified button on the NXT is pressed.

Command:

```
untilButtonPress(lcdButton);
```

Parameters: `lcdButton`

Valid LCD Buttons for `lcdButton`:
`centerBtnNXT` - NXT orange center button
`rightBtnNXT` - NXT right button
`leftBtnNXT` - NXT left button

Usage without Parameters:

```
forward();
untilButtonPress();
stop();
```

This snippet of code will run the robot forward until a button on the NXT is pressed. The default button is `centerBtnNXT` for `untilBtnPress()`.

Usage with Parameters:

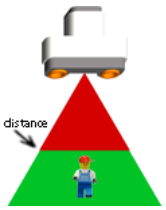
```
forward(50);
untilButtonPress(rightBtnNXT);
stop();
```

This snippet of code will run the robot forward at half speed until the right button on the NXT is pressed.

ROBOTC Natural Language - TETRIX Reference:

Until Sonar Greater Than

The robot continues what it was doing until the sonar sensor reads a value greater than a set distance in centimeters.



Command:

```
untilSonarGreaterThan(distance, sensorPort);
```

Parameters: `distance, sensorPort`

Acceptable Values for `distance`:
0 to 647 (cm).

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilSonarGreatherThan();
stop();
```

This snippet of code will run the robot forward until the sonar sensor in sensor-port 4 reads a value greater than 30 centimeters, and then stop. The default distance and sensor ports are **`30`** and **`S4`** for **`untilSonarGreaterThan()`**.

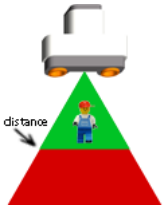
Usage with Parameters:

```
forward(50);
untilSonarGreatherThan(45, S2);
stop();
```

This snippet of code will run the robot forward at half speed until the sonar sensor in sensor-port 2 reads a value greater than 45 centimeters, and then stop.

Until Sonar Less Than

The robot continues what it was doing until the sonar sensor reads a value less than a set distance in centimeters.



Command:

```
untilSonarLessThan(distance, sensorPort);
```

Parameters: `distance, sensorPort`

Acceptable Values for `distance`:
0 to 647 (cm).

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilSonarLessThan();
stop();
```

This snippet of code will run the robot forward until the sonar sensor in sensor-port 4 reads a value less than 30 centimeters, and then stop. The default distance and sensor ports are **`30`** and **`S4`** for **`untilSonarLessThan()`**.

Usage with Parameters:

```
forward(50);
untilSonarLessThan(45, S2);
stop();
```

This snippet of code will run the robot forward at half speed until the sonar sensor in sensor-port 2 reads a value less than 45 centimeters, and then stop.

ROBOTC Natural Language - TETRIX Reference:

Until Dark

The robot continues what it was doing until the line tracking sensor reads a value darker than a specified threshold.

Command:

```
untilDark(threshold, sensorPort);
```

Parameters: `threshold`, `sensorPort`

Valid Range Values for `threshold`:
(dark) 0 to 100 (light)

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilDark();
stop();
```

This snippet of code will run the robot forward until the light sensor in sensor-port 3 reads a value darker than 45, and then stop. The default threshold and sensor port are `45` and `S3` for `untilDark()`.

Usage with Parameters:

```
forward(50);
untilDark(75, S1);
stop();
```

This snippet of code will run the robot forward at half speed until the light sensor in sensor-port 1 reads a value darker than 75, and then stop.

Until Light

The robot continues what it was doing until the line tracking sensor reads a value lighter than a specified threshold.

Command:

```
untilLight(threshold, sensorPort);
```

Parameters: `threshold`, `sensorPort`

Valid Range Values for `threshold`:
(dark) 0 to 100 (light)

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilLight();
stop();
```

This snippet of code will run the robot forward until the light sensor in sensor-port 3 reads a value lighter than 45, and then stop. The default threshold and sensor port are `45` and `S3` for `untilDark()`.

Usage with Parameters:

```
forward(50);
untilLight(15, S1);
stop();
```

This snippet of code will run the robot forward at half speed until the light sensor in sensor-port 1 reads a value lighter than 15, and then stop.

ROBOTC Natural Language - TETRIX Reference:

Until Sound Greater Than

The robot continues what it was doing until the sound sensor reads a value greater than a set threshold level.

Command:

```
untilSoundGreaterThan(threshold, sensorPort);
```

Parameters: `threshold, sensorPort`

Acceptable Values for `threshold`:
(quiet) 0 to 100 (loud).

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilSoundGreatherThan();
stop();
```

This snippet of code will run the robot forward until the sound sensor in sensor-port 2 reads a value greater than 50, and then stop. The default threshold and sensor ports are `50` and `S2` for `untilSoundGreaterThan()`.

Usage with Parameters:

```
forward(50);
untilSoundGreatherThan(85, S3);
stop();
```

This snippet of code will run the robot forward at half speed until the sound sensor in sensor-port 3 reads a value greater than (louder than) 85, and then stop.

Until Sound Less Than

The robot continues what it was doing until the sound sensor reads a value less than a set threshold level.

Command:

```
untilSoundLessThan(threshold, sensorPort);
```

Parameters: `distance, sensorPort`

Acceptable Values for `threshold`:
(quiet) 0 to 100 (loud).

Acceptable Sensors for `sensorPort`:
SENSOR ports 2 through 4 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilSoundLessThan();
stop();
```

This snippet of code will run the robot forward until the sound sensor in sensor-port 2 reads a value quieter than 50, and then stop. The default threshold and sensor ports are `50` and `S2` for `untilSoundLessThan()`.

Usage with Parameters:

```
forward(50);
untilSoundLessThan(15, S3);
stop();
```

This snippet of code will run the robot forward at half speed until the sound sensor in sensor-port 3 reads a value less than (quieter than) 15, and then stop.

ROBOTC Natural Language - TETRIX Reference:

Until Rotations

The robot continues what it was doing until the motor encoder rotations reach the desired value.

Command:

```
untilRotations(rotations, motorEncoderPort);
```

Parameters: `rotations`, `sensorPort`

Valid Range Values for `rotations`:
0.0 to 65000.0 and up.

Acceptable Sensors for `sensorPort`:
MOTOR ports A through K (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilRotations();
stop();
```

This snippet of code will run the robot forward for 1.0 rotations using a built-in encoder on motor-port D, and then stop. The default rotations and sensor port are `1.0` and `motorD` for `untilRotations()`.

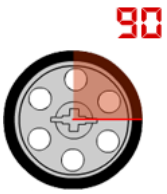
Usage with Parameters:

```
forward(50);
untilRotations(2.75, motorF);
stop();
```

This snippet of code will run the robot forward at half speed for 2.75 rotations using a built-in encoder on motor-port F, and then stop.

Until Encoder Counts

The robot continues what it was doing until the motor encoder counts reach the desired value.



Command:

```
untilEncoderCounts(counts, sensorPort);
```

Parameters: `counts`, `sensorPort`

Valid Range Values for `counts`:
0 to 65000 and up.

Acceptable Sensors for `sensorPort`:
MOTOR ports A through K (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilEncoderCounts();
stop();
```

This snippet of code will run the robot forward for 360 encoder counts (1.0 rotations) using a built-in encoder on motor-port D, and then stop. The default rotations and sensor port are `360` and `motorD` for `untilEncoderCounts()`.

Usage with Parameters:

```
forward(50);
untilEncoderCounts(990, motorF);
stop();
```

This snippet of code will run the robot forward at half speed for 990 encoder counts (2.75 rotations) using a built-in encoder on motor-port F, and then stop.