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.)
- recbot - sets the motors and sensors to match a default Recbot.
- swervebot - sets the motors and sensors to match a default Swervebot.

Usage without Parameters:

```
robotType();
```

Usage with Parameters:

```
robotType(recbot);
```

This snippet of code will set the robot type to recbot. This will automatically set up the motor and sensor ports to match those of a default Recbot.

Movement Functions:

**Set Servo**
Set a servo to a desired position.

Command:

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

Parameters: servo, position

**Acceptable Motors for servo:**
MOTOR ports 2 through 9 (and your names for them given in Motors and Sensors Setup.)

**Valid Range Values for position:**
-127 to 127.

Usage without Parameters:

```
setServo();
```

Usage with Parameters:

```
setServo(port8, 37);
```

This snippet of code will set the servo on motor-port 8 to position 37.
**Start Motor**
Set a motor to a speed.

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

**Parameters:** motor, speed

**Acceptable Motors for motor:**
MOTOR ports 1 through 10 (and your names for them given in Motors and Sensors Setup.)

**Valid Range Values for speed:**
-127 (reverse) to 127 (forward) where 0 is stop.

**Usage without Parameters:**
```
startMotor();
wait();
stopMotor();
```

This snippet of code will run the motor in motor-port 6 at speed 95 for 1.0 seconds and then stop it. The default motor-port is **port6** and the default speed is 95 for **startMotor()**.

**Usage with Parameters:**
```
startMotor(port8, -32);
wait(0.5);
stopMotor(port8);
```

This snippet of code will run the motor in motor-port 8 at speed -32 for 0.5 seconds and then stop it.

---

**Stop Motor**
Stops a motor.

Command:
```
stopMotor(motor);
```

**Parameters:** motor

**Acceptable Motors for motor:**
MOTOR ports 1 through 10 (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 6 at speed 95 for 1.0 seconds and then stop it. The default motor-port is **port6** for **stopMotor()**.

**Usage with Parameters:**
```
startMotor(port8, -32);
wait(0.5);
stopMotor(port8);
```

This snippet of code will run the motor in motor-port 8 at speed -32 for 0.5 seconds and then stop it.
Wait Functions:

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

**Command:**
```c
wait(time);
```

**Parameters:** `time`

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

**Usage without Parameters:**
```c
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:**
```c
forward(63);
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 measured in milliseconds. The robot continues to do what it was doing during this time.

**Command:**
```c
waitInMilliseconds(time);
```

**Parameters:** `time`

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

**Usage without Parameters:**
```c
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:**
```c
forward(63);
waitInMilliseconds(2730);
stop();
```

This snippet of code will run the robot forward at half speed for 2730 milliseconds (2.73 seconds) and then stop.
Robot Movement Functions:
Note that for desirable results with the following set of functions, you must use the "robotType();" Setup Function with either recbot or swervebot 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 127 (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 95 for forward().

Usage with Parameters:

```
forward(63);
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:
-127 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 -95 for backward().

Usage with Parameters:

```
backward(-63);
wait(2.0);
stop();
```

This snippet of code will run the robot backward at half speed for 2.0 seconds and then stop.
Point Turn
Both wheels rotate at the same speed but in opposite directions, causing the robot to turn in place.

Command:

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

Parameters: direction, speed

Valid Directions for direction:
- left and right.

Valid Range Values for speed:
- -127 to 127.

Usage without Parameters:

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

This snippet of code will make the robot turn right in place at speed 95 for 1.0 seconds and then stop. The default direction and speed are right and 95 for `pointTurn()`.

Usage with Parameters:

```c
pointTurn(left, 63);
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:

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

Parameters: direction, speed

Valid Directions for direction:
- left and right.

Valid Range Values for speed:
- -127 to 127.

Usage without Parameters:

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

This snippet of code will make the robot make a wide right turn at speed 95 for 1.0 seconds and then stop. The default direction and speed are right and 95 for `swingTurn()`.

Usage with Parameters:

```c
swingTurn(left, 63);
wait(0.75);
stop();
```

This snippet of code will make the robot make a wide left turn at half speed for 0.75 seconds.
**Stop**
Both wheels do not move, causing the robot to stop.

Command:

```c
stop();
```

**Parameters:** N/A

**Usage without Parameters:**

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:**

```c
forward(63);
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:

```c
lineTrackForTime(time, threshold, sensorLeft, sensorCenter, sensorRight);
```

**Parameters:** `time`, `threshold`, `sensorLeft`, `sensorCenter`, `sensorRight`

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

**Valid Range Values for `threshold`:**
(light) 0 to 4095 (dark).

**Acceptable Sensors for `sensorLeft`, `sensorCenter`, `sensorRight`:**
ANALOG ports 1 through 8 (and your names for them given in Motors and Sensors Setup.)

**Usage without Parameters:**

```c
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 505 and line tracking sensors in analog-ports `in1`, `in2`, and `in3` (L, C, R) and then stop. These values and sensors are the defaults for `lineTrackForTime()`.

**Usage with Parameters:**

```c
lineTrackForTime(7.5, 99, in6, in7, in8);
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 99 and line tracking sensors in analog-ports 6, 7, and 8 (L, C, R) and then stop.
Line Track for Rotations
The robot will track a dark line on a light surface for a specified distance in encoder rotations.

Command:

```c
lineTrackForRotations(rotations, threshold, sensorLeft, sensorCenter, sensorRight);
```

Parameters: rotations, threshold, sensorLeft, sensorCenter, sensorRight

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

Valid Range Values for threshold:
(light) 0 to 4095 (dark).

Acceptable Sensors for sensorLeft, sensorCenter, sensorRight:
ANALOG ports 1 through 8 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

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

Usage with Parameters:

```c
lineTrackForRotations(4.75, 99, in6, in7, in8);
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 99 and line tracking sensors in analog-ports 6, 7, and 8 (L, C, R) and then stop.
**Move Straight for Time**
The robot will use encoders to maintain a straight course for a specified length of time in seconds.

Command:

```c
moveStraightForTime(time, rightEncoder, leftEncoder);
```

**Parameters:** time, rightEncoder, leftEncoder

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

**Acceptable Sensors for rightEncoder, leftEncoder:**
DIGITAL ports 1 through 11 (and your names for them given in Motors and Sensors Setup.)

**Usage without Parameters:**

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

**Usage with Parameters:**

```c
moveStraightForTime(7.5, dgtl5, dgtl3);
stop();
```

This snippet of code will make the robot move forward, maintaining a straight heading for 7.5 seconds using quadrature encoders in digital-ports dgtl5+dgtl6 and dgtl3+dgtl4, and then stop. These values and sensors are the defaults for `moveStraightForTime()`.

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

Command:

```c
moveStraightForRotations(time, rightEncoder, leftEncoder);
```

**Parameters:** rotations, rightEncoder, leftEncoder

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

**Acceptable Sensors for rightEncoder, leftEncoder:**
DIGITAL ports 1 through 11 (and your names for them given in Motors and Sensors Setup.)

**Usage without Parameters:**

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

**Usage with Parameters:**

```c
moveStraightForRotations(4.75, dgtl5, dgtl3);
stop();
```

This snippet of code will make the robot move forward, maintaining a straight heading for 4.75 rotations using quadrature encoders in digital-ports 5+6 and 3+4, and then stop.
**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:

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

**Parameters:** `rightJoystick`, `leftJoystick`

**Valid Channels for `rightJoystick`, `leftJoystick`:**
Any VEXnet Remote Control channel, however Ch2 and Ch3 make the most sense for this application.

Usage without Parameters:

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

This snippet of code will remote control the robot using "tank control". The default right and left joysticks are Ch2 and Ch3, and 10 for `tankControl()`.

Usage with Parameters:

```c
while(true)
{
    tankControl(Ch1, Ch4, 5);
}
```

This snippet of code will remote control the robot using "tank control" with channel 1 as the right joystick and channel 4 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:

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

**Parameters:** `verticalJoystick`, `horizontalJoystick`

**Valid Channels for `verticalJoystick`, `horizontalJoystick`:**
Any VEXnet Remote Control channel, however Ch2+Ch1 or Ch3+Ch4 make the most sense for this application.

Usage without Parameters:

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

This snippet of code will remote control the robot using "tank control". The default vertical and horizontal joysticks are Ch2 and Ch1, and 10 for `arcadeControl()`.

Usage with Parameters:

```c
while(true)
{
    arcadeControl(Ch3, Ch4, 5);
}
```

This snippet of code will remote control the robot using "tank control" with channel 3 as the vertical joystick and channel 4 as the horizontal joystick (arcade control with the left-side joystick) with a threshold of 5.
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`:
DIGITAL ports 1 through 12 (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 digital-port 6 is pressed, and then stop.
The default sensor port is `dgtl6` for `untilTouch()`.

**Usage with Parameters:**
```
forward(63);
untilTouch(dgtl10);
stop();
```

This snippet of code will run the robot forward at half speed until the touch sensor in digital-port 10 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`:
DIGITAL ports 1 through 12 (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 digital-port 6 is released, and then stop.
The default sensor port is `dgtl6` for `untilRelease()`.

**Usage with Parameters:**
```
forward(63);
untilRelease(dgtl10);
stop();
```

This snippet of code will run the robot forward at half speed until the touch sensor in digital-port 10 is released, and then stop.
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:
DIGITAL ports 1 through 12 (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();
```

Usage with Parameters:

```
forward(63);
untilBump(dgtl10, 100);
stop();
```

This snippet of code will run the robot forward until the touch sensor in digital-port 6 is pressed in and then released out, and then stop. The default sensor port and delay time are dgtl6 and 10 for untilBump().

Until Button Press
The robot continues what it was doing until a specified button on the VEX LCD is pressed. Connect the VEX LCD to UART-port 2.

Command:

```
untilButtonPress(lcdButton);
```

Parameters: lcdButton

Valid LCD Buttons for lcdButton:
centerBtnVEX - VEX LCD center button
centerBtnVEX - VEX LCD right button
centerBtnVEX - VEX LCD left button

Usage without Parameters:

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

Usage with Parameters:

```
forward(63);
untilButtonPress(rightBtnVEX);
stop();
```

This snippet of code will run the robot forward until a button on the VEX LCD is pressed. The default button is centerBtnVEX for untilBtnPress().
**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:**
DIGITAL ports 1 through 12 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

```
forward();
untilSonarGreaterThan();
stop();
```

Usage with Parameters:

```
forward(63);
untilSonarGreaterThan(45, dgt12);
stop();
```

This snippet of code will run the robot forward until the sonar sensor in digital-port 8+9 reads a value greater than 30 centimeters, and then stop. The default distance and sensor ports are 30 and dgt18(+dgt19) for `untilSonarGreaterThan()`.

**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:**
DIGITAL ports 1 through 12 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

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

Usage with Parameters:

```
forward(63);
untilSonarLessThan(45, dgt12);
stop();
```

This snippet of code will run the robot forward until the sonar sensor in digital-port 8+9 reads a value less than 30 centimeters, and then stop. The default distance and sensor ports are 30 and dgt18(+dgt19) for `untilSonarLessThan()`.

This snippet of code will run the robot forward at half speed until the sonar sensor in digital-port 2+3 reads a value less than 45 centimeters, and then stop.
Until Potentiometer Greater Than
The robot continues what it was doing until the potentiometer sensor reads a value greater than a set position.

Command:

\[\text{untilPotentiometerGreaterThan}(\text{position}, \text{sensorPort})\];

Parameters: position, sensorPort

Valid Range Values for position:
0 to 4095 (However due to mechanical stops, you may be limited to the range of 5 to 4090.)

Acceptable Sensors for sensorPort:
ANALOG ports 1 through 8 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

\[
\begin{align*}
\text{startMotor}(); \\
\text{untilPotentiometerGreaterThan}(); \\
\text{stop}();
\end{align*}
\]

Usage with Parameters:

\[
\begin{align*}
\text{startMotor}(\text{port8}, 63); \\
\text{untilPotentiometerGreaterThan}(4000, \text{in4}); \\
\text{stop}();
\end{align*}
\]

This snippet of code will run the motor on port 8 at speed 63 until the potentiometer in analog-port 4 reaches a value greater than 4000, and then stop.

Until Potentiometer Less Than
The robot continues what it was doing until the potentiometer sensor reads a value less than a set position.

Command:

\[\text{untilPotentiometerLessThan}(\text{position}, \text{sensorPort})\];

Parameters: position, sensorPort

Valid Range Values for position:
0 to 4095 (However due to mechanical stops, you may be limited to the range of 5 to 4090.)

Acceptable Sensors for sensorPort:
ANALOG ports 1 through 8 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

\[
\begin{align*}
\text{startMotor}(); \\
\text{untilPotentiometerLessThan}(); \\
\text{stop}();
\end{align*}
\]

Usage with Parameters:

\[
\begin{align*}
\text{startMotor}(\text{port8}, 63); \\
\text{untilPotentiometerLessThan}(40, \text{in4}); \\
\text{stop}();
\end{align*}
\]

This snippet of code will run the motor on port 8 at speed 63 until the potentiometer in analog-port 4 reaches a value less than 40, and then stop.
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:
(light) 0 to 4095 (dark)

Acceptable Sensors for sensorPort:
ANALOG ports 1 through 8 (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 line tracking sensor in analog-port 2 reads a value darker than 1500, and then stop. The default threshold and sensor port are 1500 and in2 for untilDark().

Usage with Parameters:
```
forward(63);
untilDark(1005, in4);
stop();
```
This snippet of code will run the robot forward at half speed until the line tracking sensor in analog-port 4 reads a value lighter than 1005, 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:
(light) 0 to 4095 (dark)

Acceptable Sensors for sensorPort:
ANALOG ports 1 through 8 (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 line tracking sensor in analog-port 2 reads a value lighter than 1500, and then stop. The default threshold and sensor port are 1500 and in2 for untilLight().

Usage with Parameters:
```
forward(63);
untilLight(1005, in4);
stop();
```
This snippet of code will run the robot forward at half speed until the line tracking sensor in analog-port 4 reads a value lighter than 1005, and then stop.
Until Rotations
The robot continues what it was doing until the quadrature encoder rotations reach the desired value.

Command:

```
untilRotations(rotations, sensorPort);
```

Parameters: rotations, sensorPort

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

Acceptable Sensors for sensorPort:
DIGITAL ports 1 through 11 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

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

Usage with Parameters:

```
forward(63);
untilRotations(2.75, dgtl3);
stop();
```

This snippet of code will run the robot forward at half speed for 990 encoder counts (2.75 rotations) using a quadrature encoder in digital-port 3+4, and then stop.

Until Encoder Counts
The robot continues what it was doing until the quadrature 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:
DIGITAL ports 1 through 11 (and your names for them given in Motors and Sensors Setup.)

Usage without Parameters:

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

Usage with Parameters:

```
forward(63);
untilEncoderCounts(990, dgtl3);
stop();
```

This snippet of code will run the robot forward at half speed for 990 encoder counts (2.75 rotations) using a quadrature encoder in digital-port 3+4, and then stop.
LED ON
Turn an LED in a specified digital-port ON.

Command:

```
turnLEDOn(sensorPort);
```

Parameters: sensorPort

Acceptable Sensors for sensorPort:
DIGITAL ports 1 through 12 (and your names for them given in Motors and Sensors Setup.)
Note that you must set these digital-ports to "VEX LED".

Usage without Parameters:
```
turnLEDOn();
```
This snippet of code will turn an LED in digital-port 2 ON. The default sensor port is dgtl2 for turnLEDOn().

Usage with Parameters:
```
turnLEDOn(dgtl7);
```
This snippet of code will turn an LED in digital-port 7 ON.

LED OFF
Turn an LED in a specified digital-port OFF.

Command:

```
turnLEDOff(sensorPort);
```

Parameters: sensorPort

Acceptable Sensors for sensorPort:
DIGITAL ports 1 through 12 (and your names for them given in Motors and Sensors Setup.)
Note that you must set these digital-ports to "VEX LED".

Usage without Parameters:
```
turnLEDOff();
```
This snippet of code will turn an LED in digital-port 2 OFF. The default sensor port is dgtl2 for turnLEDOff().

Usage with Parameters:
```
turnLEDOff(dgtl7);
```
This snippet of code will turn an LED in digital-port 7 OFF.
**Flashlight ON**

Turn a VEX Flashlight in a specified motor-port ON at a specified brightness.

**Command:**

```text
turnFlashlightOn(motorPort, brightness);
```

**Parameters:** motorPort, brightness

**Acceptable Motors for motorPort:**

MOTOR ports 1 through 10 (and your names for them given in Motors and Sensors Setup.)

*NOTE* Brightness control only available in motor-ports 1 and 10, or 2 through 9 when connected to a VEX Motor Controller 29.

**Valid Range Values for brightness:**

(off) 0 to 127 (bright)

**Usage without Parameters:**

```text
turnFlashlightOn();
```

This snippet of code will turn a VEX Flashlight in motor-port 4 ON at brightness level 63 (half bright). The default motor port and brightness are port4 and 63 for turnFlashlightOn().

**Usage with Parameters:**

```text
turnFlashlightOn(port10, 127);
```

This snippet of code will turn a VEX Flashlight in motor-port 10 ON at brightness level 127 (full bright).

---

**Flashlight OFF**

Turn a VEX Flashlight in a specified motor-port OFF.

**Command:**

```text
turnFlashlightOff(motorPort);
```

**Parameters:** motorPort

**Acceptable Motors for motorPort:**

MOTOR ports 1 through 10 (and your names for them given in Motors and Sensors Setup.)

**Usage without Parameters:**

```text
turnFlashlightOff();
```

This snippet of code will turn a VEX Flashlight in motor-port 4 OFF. The default motor port is port4 OFF.

**Usage with Parameters:**

```text
turnFlashlightOff(port10);
```

This snippet of code will turn a VEX Flashlight in motor-port 10 OFF.