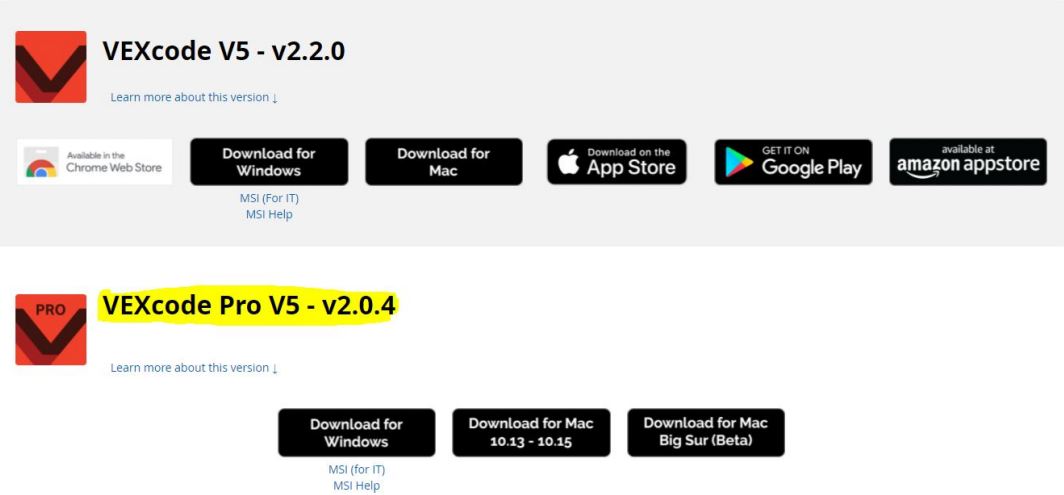
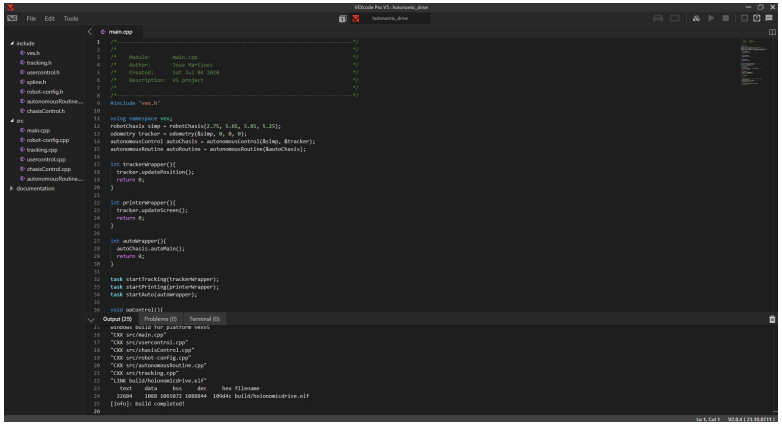
Matabots
Programming Workshop

CSUN VEX Robotics

VEXcode Pro V5

Download VEXcode Pro V5





https://www.vexrobotics.com/vexcode-download

C++ VEXcode Program

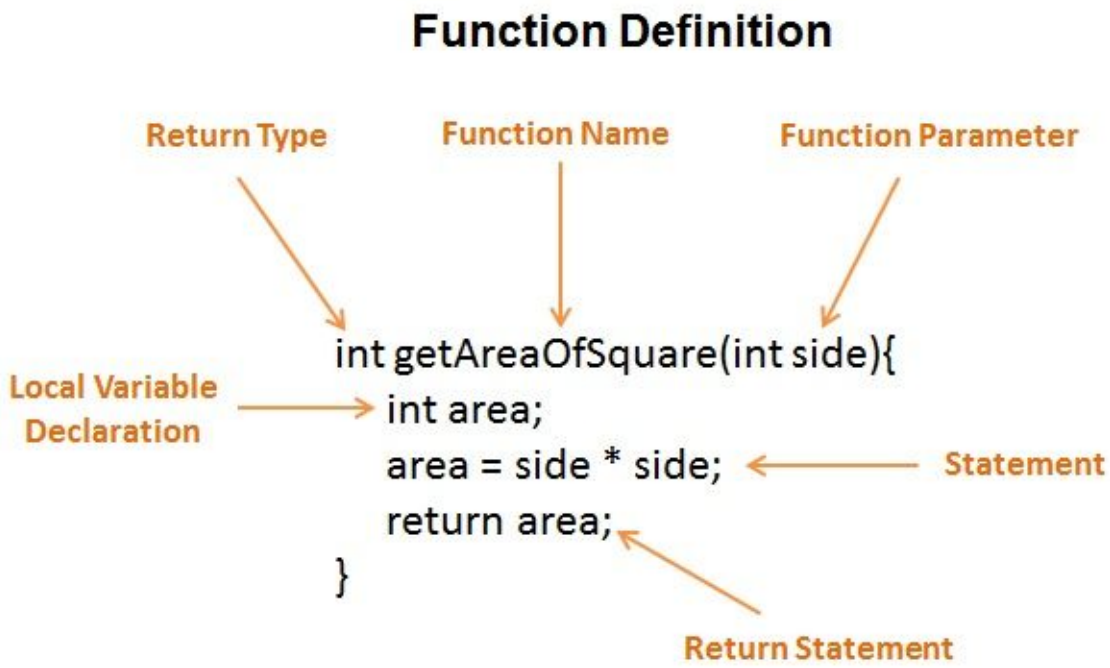
main.cpp

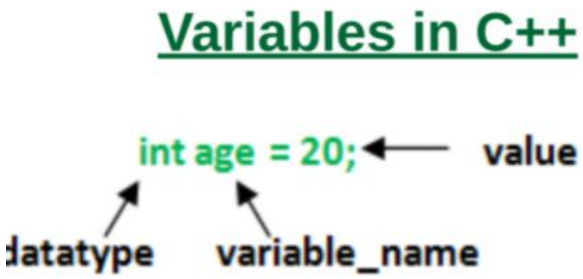
```cpp
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*    Module:       main.cpp                                                  */
/*    Author:       C:\Users\mrlop                                            */
/*    Created:      Fri Oct 22 2021                                           */
/*    Description:  V5 project                                                */
/*                                                                            */
/*----------------------------------------------------------------------------*/

// ---- START VEXCODE CONFIGURED DEVICES ----
// ---- END VEXCODE CONFIGURED DEVICES ----

#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

}

```

Basic C++ Program

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout<<"Hello World";

    return 0;
}

```

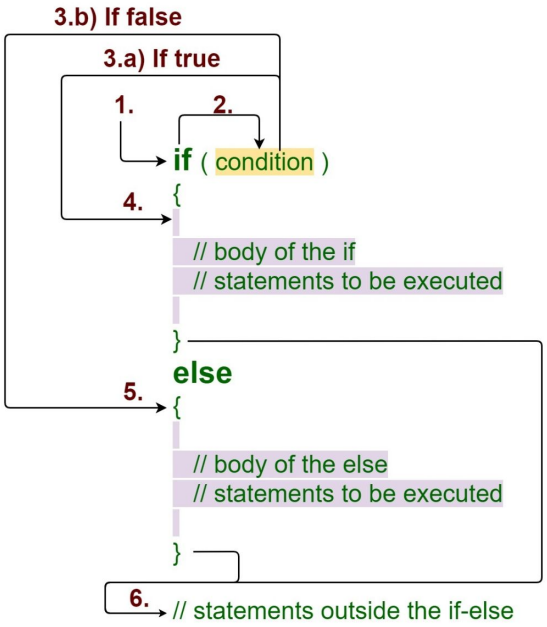
Basic C++ syntax





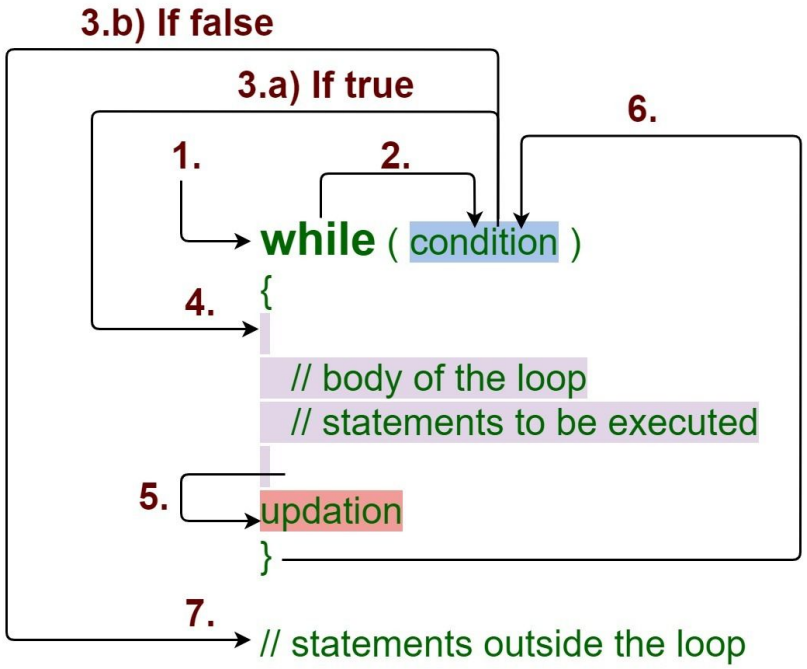
Basic C++ syntax

If - else statement

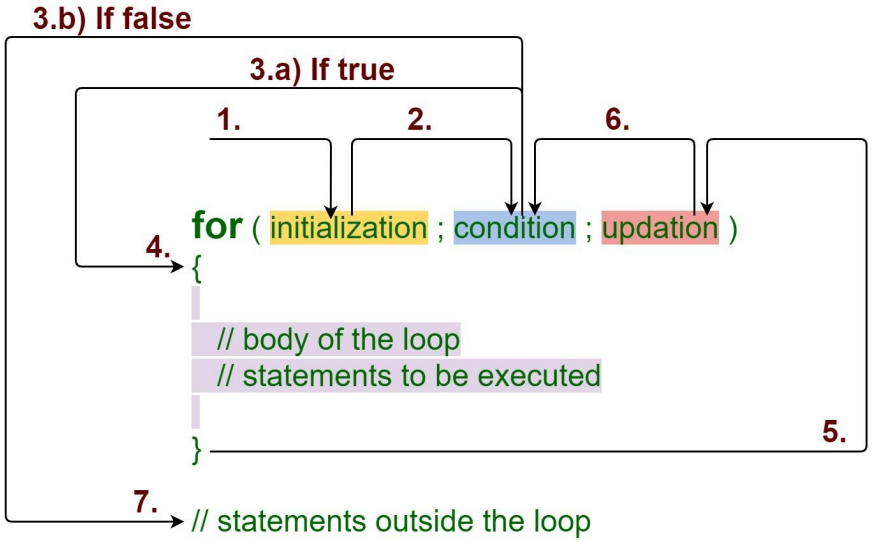


Basic C++ syntax

While Loop



For Loop

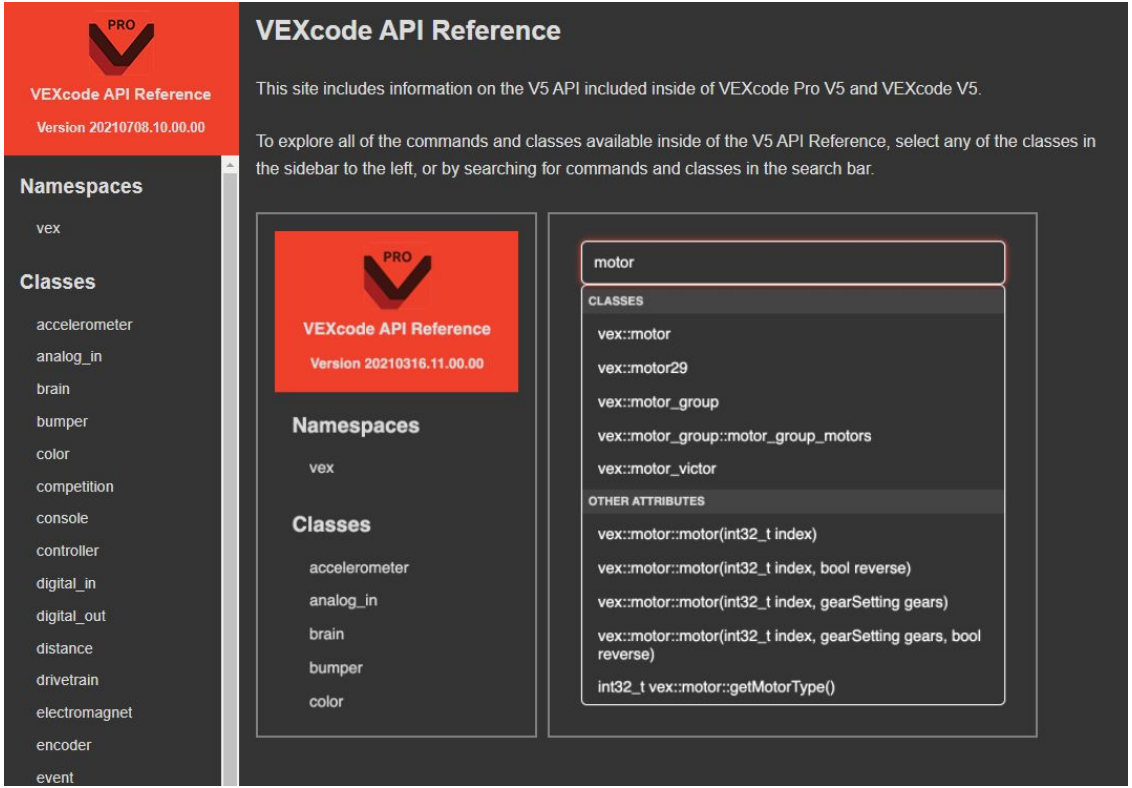


Basic C++ syntax

```cpp
class Box {
    public:
        double length;          // Length of a box
        double breadth;         // Breadth of a box
        double height;          // Height of a box

        // Member functions declaration
        double getVolume(void);
        void setLength( double len );
        void setBreadth( double bre );
        void setHeight( double hei );
};

// Member functions definitions
double Box::getVolume(void) {
    return length * breadth * height;
}

void Box::setLength( double len ) {
    length = len;
}
void Box::setBreadth( double bre ) {
    breadth = bre;
}
void Box::setHeight( double hei ) {
    height = hei;
}

```

```cpp
class Box {
    public:
        double length;      // Length of a box
        double breadth;     // Breadth of a box
        double height;      // Height of a box

        double getVolume(void) {
            return length * breadth * height;
        }
};
int main() {
    // Initializing Robot Configuration. DO NOT REMOVE!
    vexcodeInit();

    Box Box1;               // Declare Box1 of type Box
    Box Box2;               // Declare Box2 of type Box
    double volume = 0.0;    // Store the volume of a box here

    // box 1 specification
    Box1.setLength(6.0);
    Box1.setBreadth(7.0);
    Box1.setHeight(5.0);

    // box 2 specification
    Box2.setLength(12.0);
    Box2.setBreadth(13.0);
    Box2.setHeight(10.0);

    // volume of box 1
    volume = Box1.getVolume();
    printf("Volume of Box1 : ");

    // volume of box 2
    volume = Box2.getVolume();
    printf("Volume of Box2 : ");
}
```

VEXcode API

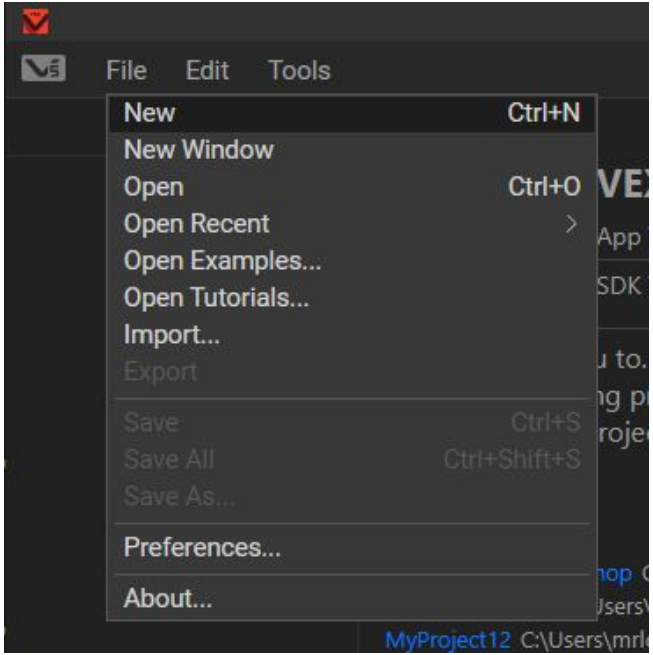


Your best friend!

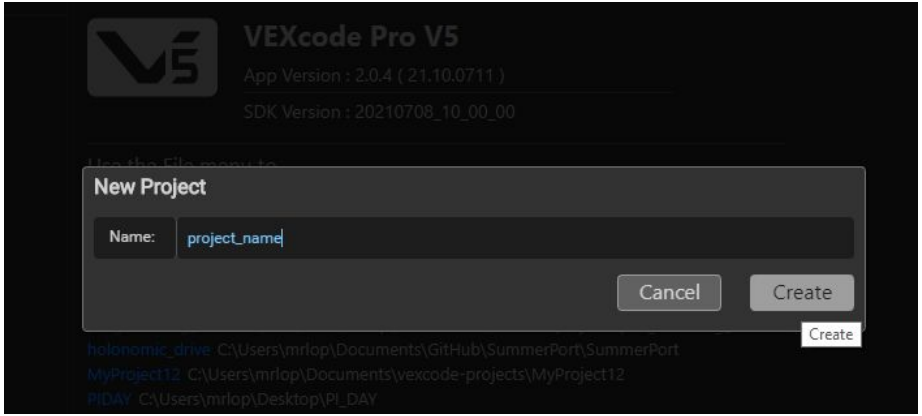
https://api.vexcode.cloud/v5/

Creating a VEXcode Pro V5 Project

1.



2.



.h file vs .cpp file

- include
  - robot-config.h
  - vex.h
- src
  - main.cpp
  - robot-config.cpp

What are these files and how are they different?

robot-config.h

```
using namespace vex;

extern brain Brain;
extern motor frontRight;
/**
 * Used to initialize code/tasks/devices added using tools in VEXcode Pro.
 *
 * This should be called at the start of your int main function.
 */
void vexcodeInit(void);

```

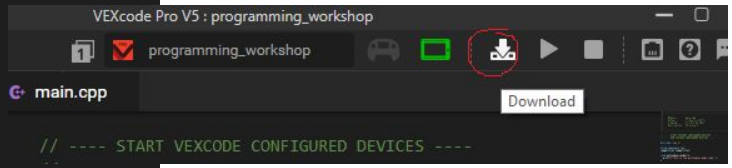
robot-config.cpp

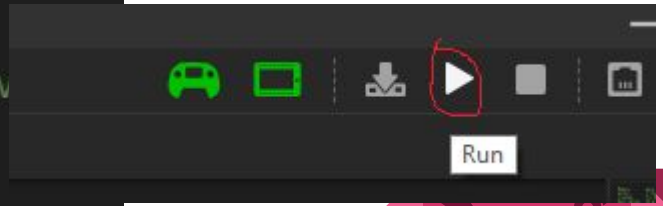
```
#include "vex.h"

using namespace vex;

// A global instance of brain used for printing to the V5 brain screen
brain Brain;
motor frontRight = motor(PORT8, ratio18_1, false);
/**
 * Used to initialize code/tasks/devices added using tools in VEXcode Pro.
 *
 * This should be called at the start of your int main function.
 */
void vexcodeInit(void) {
  // Nothing to initialize
}
```

Uploading & Running your VEXcode Pro V5 Project

Modify the main.cpp file.

```
19  void driver_control(){
20    printf("This is driver control.");
21  }
22
23  void autonomous_routine(){
24    printf("This is the autnomous routine.");
25  }
26
27  int main() {
28    // Initializing Robot Configuration. DO NOT REMOV
29    vexcodeInit();
30    Competition.autonomous(driver_control);
31    Competition.drivercontrol(autonomous_routine);
32  }
33
```





V5 Brain

V5 Brain: Controls all logic for the robot. Your program is uploaded & runs here.

Controller: Programmed to control the robot. User controls are uploaded and ran on the device.

Radio: Provides communication between the controller & the V5 brain.

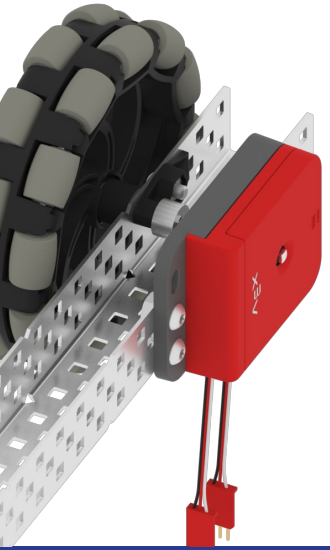
V5 Cable: For Pins 1-21

3 Wire Cable: For Pins A-H

3 Wire Cable

Sensors

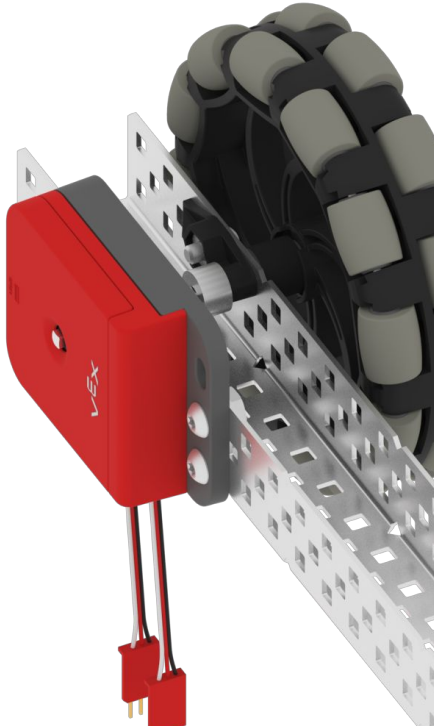
- Sensors are what the robot will use to gather information about its surroundings.
- They are really important in controlling actuators.
- They help the robot decide on performing actions or how much of an action to do.



Rotation Sensor/Encoder

Operated by detecting the amount the shaft in the encoder has rotated.

- Rotation sensor has a 0.088° accuracy.
- Encoder sensor has a 1° accuracy.



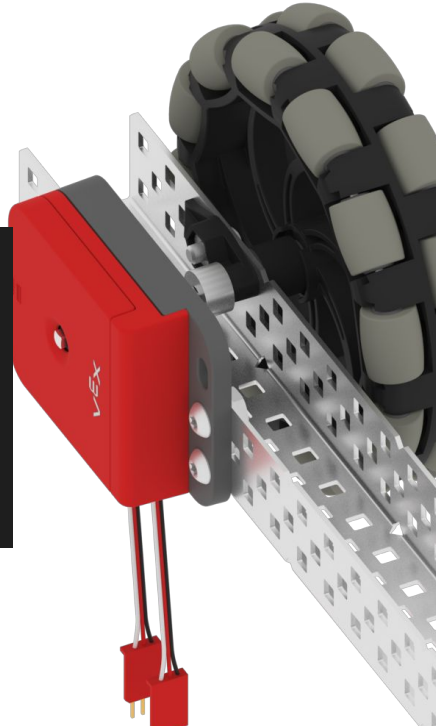
Rotation Sensor/Encoder

Code (Encoder)

```
void autonomous_routine(){
  printf("This is the autnomous routine.");
  // Creating an encoder object.
  encoder encoder_example = encoder(Brain.ThreeWirePort.A);

  // Read current encoder value.
  int encoder_value = encoder_example.rotation(deg);

  // Print encoder value.
  printf("Encoder Value = %d", encoder_value);
}
```

Code (Rotation)

```
void autonomous_routine(){
  printf("This is the autnomous routine.");
  // Creating an rotation object.
  rotation rotation_example = rotation(15);

  // Read current rotation value.
  int rotation_value = rotation_example.position(deg);

  // Print rotation value.
  printf("rotation Value = %d", rotation_value);
}
```



rotationUnits

The measurement units for rotation values.

Values	
deg	A rotation unit that is measured in degrees.
rev	A rotation unit that is measured in revolutions.
raw	A rotation unit that is measured in raw data form.

Try on your own!

vex::rotation

Use the rotation class to control the V5 Rotation Sensor.

rotation::rotation(int32_t index, bool reverse=false)
The constructor for the Rotation sensor.

bool rotation::installed()

void rotation::setReversed(bool value)
Gets the absolute angle of the Rotation sensor.

double rotation::angle(rotationUnits units=rotationUnits::deg)
Resets the position of the Rotation sensor to the value of 0.

void rotation::resetPosition(void)
Sets the position value of the Rotation sensor to the specified value and units defined ...

void rotation::setPosition(double value, rotationUnits units)
Gets the current position of the Rotation sensor.

double rotation::position(rotationUnits units)
Gets the current velocity of the Rotation sensor.

double rotation::velocity(velocityUnits units)
Calls the specified function when the Rotation value changes.

void rotation::changed(void(*callback)(void))
Calls a function when the encoder value changes.

vex::encoder

Use this class when programming with an Encoder.

encoder::encoder(triport::port &port)
The constructor for the Encoder.

void encoder::resetRotation(void)

void encoder::setRotation(double val, rotationUnits units)
Gets the rotation value of the Encoder.

void encoder::setPosition(double val, rotationUnits units)

double encoder::rotation(rotationUnits units)
Gets the velocity of the Encoder.

double encoder::position(rotationUnits units)
Calls a function when the Encoder value changes.

double encoder::velocity(velocityUnits units)
Gets the velocity of the encoder.

void encoder::changed(void(*callback)(void))
Calls a function when the encoder value changes.

Try to continually read and print then sensor values to the console.

Hint: **Use printf("Rotation Value : %d", rotation_value);**

MATABOTS

Limit Switch

Operated by detecting the presence or absence of an object.

Code:

```
void autonomous_routine(){
  printf("This is the autnomous routine.");
  // Creating a limit object.
  limit limit_example = limit(Brain.ThreeWirePort.A);

  // Read current limit value.
  int limit_value = limit_example.pressing();

  // Print limit value.
  printf("Limit Value = %d", limit_value);
}
```

Try on your own!

vex::limit

Use this class when programming with the Limit Switch.

limit::limit(triport::port &port)
The constructor for the Limit Switch.

int32_t limit::pressing()
Runs the callback function when the Limit switch is pressed.

void limit::pressed(void(*callback)(void))
Runs the callback function when the Limit switch is released.

void limit::released(void(*callback)(void))
Calls a function when the limit switch is released.

Try to print **“I love CSUN Matabots! <3”** 🥰 whenever the limit switch is pressed.

Hint: Use a while loop that checks if the limit switch is pressed.

Optical

Converts light/color into a proximity and gesture sensor.

Code:

```
void autonomous_routine(){
  printf("This is the autnomous routine.");
  // Creating a limit object.
  optical optical_example = (10);

  // Read current limit value.
  optical::rgbc optical_value = optical_example.getRgb();

  // Print limit value.
  printf("Optical Values: Red:  %lf, Green: %lf, and Blue: %lf",
  optical_value.red, optical_value.green, optical_value.blue);
}
```

Try on your own!

vex::optical

Use the optical class to control the V5 Optical sensor.

optical::optical(int32_t index, bool enableGesture=false)
The constructor for the Optical sensor.

bool optical::installed()

double optical::hue()
Gets the brightness value detected by the Optical sensor.

double optical::brightness(bool bRaw=false)
Gets the color of a detected object.

color vex::optical::color()
Tells whether an object is within range of the Optical sensor.

bool optical::isNearObject()
Runs the callback function when an object is detected by the Optical sensor.

void optical::objectDetected(void(*callback)(void))
Runs the callback function when an object is no longer detected by the Optical sensor.

void optical::objectLost(void(*callback)(void))
Sets the threshold for object detection.

int32_t optical::objectDetectThreshold(int32_t value=0)
Runs the callback function when the Optical sensor detects upwards movement.

void optical::gestureUp(void(*callback)(void))
Runs the callback function when the Optical sensor detects downward movement.

void optical::gestureDown(void(*callback)(void))
Runs the callback function when the Optical sensor detects movement to the left.

void optical::gestureLeft(void(*callback)(void))
Runs the callback function when the Optical sensor detects movement to the right.

void optical::gestureRight(void(*callback)(void))
Sets the state of the Optical Sensor's LED.

void optical::setLight(ledState state)
Set the intensity of the led on the Optical sensor in percent.

void optical::setLightPower(int32_t intensity, percentUnits units=percentUnits::pct)
Enables gesture detection on the Optical sensor.

void optical::gestureEnable(void)
Disables gesture detection on the Optical sensor.

void optical::gestureDisable(void)
Get object with the most recent gesture detection data.

Try to turn on the LED.
Try to continually print Brightness and RGB values.

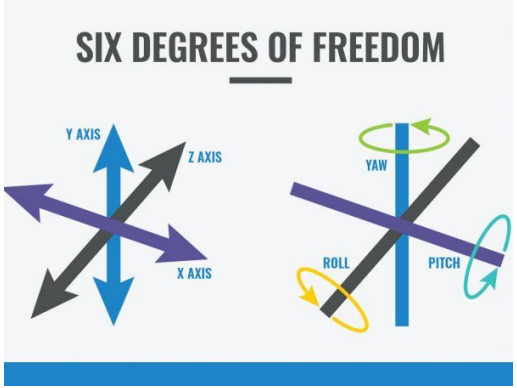
Hint: Use brightness() to read the brightness value.

MATABOTS

IMU

Measures angular velocity and acceleration.

Can be used to determine orientation and movement of the robot.



Code:

```
void autonomous_mode(){
  printf("This is the autonomous mode code.");

  // Create an IMU pbject.
  inertial inertial_example = inertial(10);

  // Read heading value from the IMU.
  double heading_value = inertial_example.heading();

  // Print heading value to the console
  printf("Heading value: %lf", heading_value);
}
```

Try on your own!

vex::inertial

Use the inertial class to control the V5 Inertial Sensor.

inertial::inertial(int32_t index)
The constructor for the Inertial sensor.

bool inertial::installed()

void inertial::startCalibration(int32_t value=0)
Calibrates the Inertial sensor for the given time.

void inertial::calibrate(int32_t value=0)
Checks if the Inertial sensor is currently calibrating.

bool inertial::isCalibrating(void)
Resets the Inertial sensor heading to 0.

void inertial::resetHeading()
Resets the Inertial sensor rotation to 0.

void inertial::resetRotation()
Sets the Inertial sensor heading to the given value.

void inertial::setHeading(double value, rotationUnits units)
Set the Inertial sensor rotation to angle.

void inertial::setRotation(double value, rotationUnits units)
Gets the angle (yaw angle) of the Inertial sensor.

double inertial::angle(rotationUnits units=rotationUnits::deg)
Gets the roll angle of the Inertial sensor.

double inertial::roll(rotationUnits units=rotationUnits::deg)
Gets the pitch angle of the Inertial sensor.

double inertial::pitch(rotationUnits units=rotationUnits::deg)
Gets the yaw angle of the Inertial sensor.

double inertial::yaw(rotationUnits units=rotationUnits::deg)
Gets an orientation angle of the Inertial sensor.

double inertial::orientation(orientationType axis, rotationUnits units)
Gets the heading (yaw angle as 0-360 deg) of the Inertial sensor.

double inertial::heading(rotationUnits units=rotationUnits::deg)
Gets the absolute angle (yaw angle without limits) of the Inertial sensor.

double inertial::rotation(rotationUnits units=rotationUnits::deg)
Sets the Inertial sensor orientation in quaternion form.

void inertial::orientation(quaternion &q)
Gets the Inertial sensor orientation in quaternion form.

quaternion inertial::orientation()
Sets the Inertial sensor orientation in rotation form.

void inertial::orientation(attitude &a)
Gets the Inertial sensor raw gyro data in specified units.

double inertial::gyroRate(axisType axis, velocityUnits units)
Gets the Inertial sensor raw acceleration data in G.

double inertial::acceleration(axisType axis)
Calls a function when the Inertial sensor heading value changes.

void inertial::changed(void(*callback)(void))
Calls a function when the Inertial sensor detects a collision.

void inertial::collision(void(*callback)(axisType, double, double, double))
Calls a function when the inertial sensor detects a collision.

Try to continually print heading and acceleration for the x, y, and z axis.

Hint: **Use inertial_example.()**

Line Sensor

Reads the difference in light on surfaces to to determine an outcome.

Example: Reads a white line on a black surface.

- Can be used to maintain a robot's position.

Line Sensor

- Code:

```
void autonomous_mode(){
  printf("This is the autonomous mode code.");

  // Create a line sensor object.
  line line_example = line(Brain.ThreeWirePort.A);

  // Read line sensor value from the IMU.
  int line_reflectivity = line_example.reflectivity();

  // Print lien sensor value to the console
  printf("Line sensor value: %d", line_reflectivity);
}
```

Try on your own!

vex::line

Use this class when programming with a Line Tracker sensor.

line::line(triport::port &port)
The constructor for the Line Tracker sensor.

int32_t line::reflectivity(percentUnits units=percentUnits::pct)
Calls a function when the Line Tracker sensor value changes.

void line::changed(void(*callback)(void))
Calls a function when the line sensor value changes.

Try to print “I love CSUN Matabots <3” whenever the limit switch .

Hint: **Use printf(“Rotation Value : %d”, rotation_value);**

Motors

Next week

Try on your own!

vex::line

Use this class when programming with a Line Tracker sensor.

line::line(triport::port &port)
The constructor for the Line Tracker sensor.

int32_t line::reflectivity(percentUnits units=percentUnits::pct)
Calls a function when the Line Tracker sensor value changes.

void line::changed(void(*callback)(void))
Calls a function when the line sensor value changes.

Try to print “I love CSUN Matabots <3” whenever the limit switch .

Hint: **Use printf(“Rotation Value : %d”, rotation_value);**

Controller

Try on your own!

vex::controller

Use the controller class to get values from the remote controller as well as write to the controller's screen.

button& controller::ButtonL1
A button that represents the L1 button on the controller.

button& controller::ButtonL2
A button that represents the L2 button on the controller.

button& controller::ButtonR1
A button that represents the R1 button on the controller.

button& controller::ButtonR2
A button that represents the R2 button on the controller.

button& controller::ButtonUp
A button that represents the up button on the controller.

button& controller::ButtonDown
A button that represents the down button on the controller.

button& controller::ButtonLeft
A button that represents the left button on the controller.

button& controller::ButtonRight
A button that represents the right button on the controller.

button& controller::ButtonX
A button that represents the x button on the controller.

button& controller::ButtonB
A button that represents the B button on the controller.

button& controller::ButtonY
A button that represents the Y button on the controller.

button& controller::ButtonA
A button that represents the A button on the controller.

axis& controller::Axis1
An axis of a joystick that represents axis 1 on the controller.

axis& controller::Axis2
An axis of a joystick that represents axis 2 on the controller.

axis& controller::Axis3
An axis of a joystick that represents axis 3 on the controller.

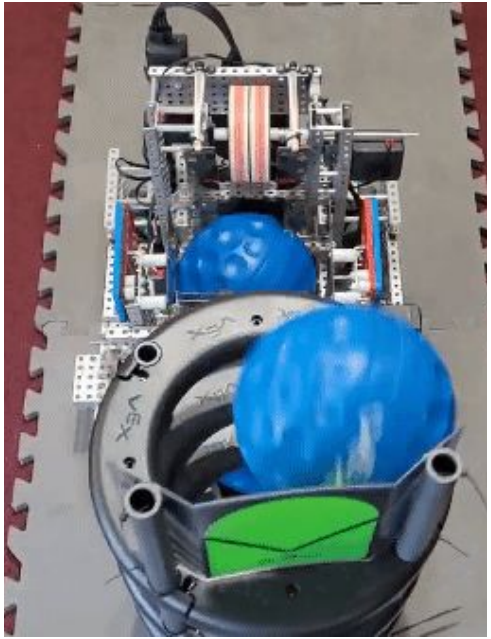
axis& controller::Axis4
An axis of a joystick that represents axis 4 on the controller.

lcd controller::Screen
Controller's instance of LCD.

Try to print “I love CSUN Matabots <3” whenever the limit switch .

Hint: **Use printf(“Rotation Value : %d”, rotation_value);**

Autonomous



VEX API