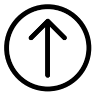
COD-04

C-Language Fundamentals

EXCEED
ROBOTICS



EXCEED ROBOTICS

STUDENT NAME

© 2024 Exceed Robotics ™

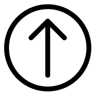


TABLE OF CONTENTS

About this workbook .... **i**

Overview i

Using this workbook i

Week 1: Basic movement **1**

What is Programming? 1

Getting started 2

Understanding the components of the robot 2

Robot setup for this course 3

Creating your first program 4

Understanding The RobotC Layout 6

Basic Commands 8

Driving basics 8

Example: Driving basics 11

Using the forklift 11

Example: Using the forklift 13

Syntax – The Rules of Programming 14

What is Syntax? 14

Example: Finding Errors 16

In class exercises 17

Homework 19

What is the command? 19

What does it all mean? 20

Complete the code 21

Converting From Instructions to Code 22

Week 2 – Variables **23**

What are variables? 23

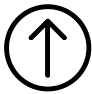
Types of variables 23

Example: Types of Variables 24

How to create variables? 24

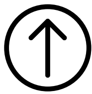
Example: How to create variables 24

© 2024 Exceed Robotics ™



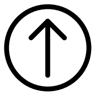
Visualizing computer memory 25
Using variables you create in your program 28
Updating your variables inside your code 32
Syntax for variables 35
In-Class Exercises 38
Homework 42
Check your understanding 42
Converting from instructions to Code 43
Fill in the memory box 45
Week 3 & 4 – If Statements 47
Display text on the Ev3 screen 47
What is conditional programming? 48
How do we write Boolean expressions? 48
Evaluating Boolean expressions 49
Boolean expressions with variables 50
What is an IF statement? 52
The structure of an IF statement 52
How does an IF Statement Work (Example 1) 53
How does an IF Statement Work (Example 2) 56
Example: Writing If Statements 58
Using the EV3 Buttons 59
Example using a button and IF statements. 61
Syntax for IF statements 62
In Class problems 65
Homework 68
Check your understanding 68
Coding Exercises 69
Week 5 – While loops 71
What are while loops? 71
Structure of While Loops 71
How do while loops work? 71
In-Class Exercises 81

© 2024 Exceed Robotics ™



Homework....84
Coding Exercises....84
Check your understanding....85
WEEK 6, 7 and 8 - Project....87
Introduction to the project....87
Background....87
Task 1 – Basic Program....87
Task 2 – Time-Saver!....88
Task 3 – Crate counters....88

© 2024 Exceed Robotics ™



ABOUT THIS WORKBOOK

OVERVIEW

This course booklet is designed to supplement the **RobotC**-Language classes at Exceed Robotics, by providing reference materials, examples, and homework problems for extra practice. Students are encouraged to use this booklet as a reference and complete the homework problems to both further their understanding of the programming languages as well as develop their programming skills.

Each week's material is divided into three sections.

1. Class Summary
2. In class problems
3. Homework

The class summary section offers students an opportunity to review the core topics covered in the weekly classes.

The In-class problems section describes the different problems the student should have solved during the class.

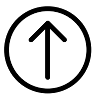
The Homework section provides additional problems that can be used to assess the student's understanding of the material. The section consists of 4 different types of problems such as,

1. Multiple choice questions
2. Troubleshooting problems
3. Task-based problems
4. Projects

USING THIS WORKBOOK

One feature of this workbook is that it is editable. The homework section contains multiple-choice questions, which require you to select the correct checkboxes. It also contains problems that require you to type in their answers. Therefore, you must save your handbook so that you do not lose your answers. To save your workbook at any time, you can use **Ctrl+S** for Windows or **Cmd+S** for MacOS. When exiting, the handbook, you will also be prompted to save any unsaved changes

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

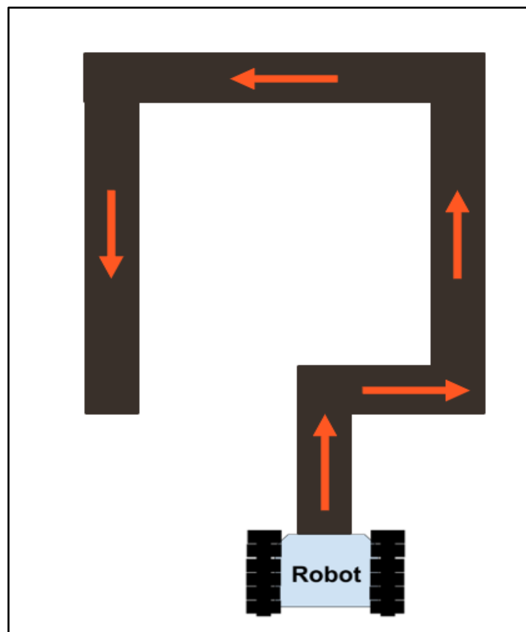
WEEK 1: BASIC MOVEMENT

WHAT IS PROGRAMMING?

Programming (or coding) is the process of creating sets of instructions that tell a computer what to do. These instructions are written in a **programming language**, which is like a special type of language that computers understand. Programmers use these instructions to create software, apps, websites, and other digital tools that we use every day. In this course, we are going to learn how to program robots.

To become good programmers, we need to think like computers and machines. You can think of computers as very obedient pets. They will try to execute all the instructions that you give them exactly [the way that you give them], so we need to be very careful about what we tell them to do.

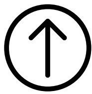
Let's look at an example now. What sort of information would you give a robot so it follows the path shown? How do we do it?



The commands we could give our robot are,

Step 1. **Go Straight**
Step 2. **Turn Right 90 degrees**
Step 3. **Go Straight**
Step 4. **Turn Left 90 degrees**
Step 5. **Go Straight**
Step 6. **Turn Left 90 degrees**
Step 7. **Go Straight**
Step 8. **Turn Left 90 degrees**
Step 9. **Go Straight**

© 2024 Exceed Robotics ™



Congratulations! You have just completed your first program. There are several programming languages available (more than a thousand!). For this course, we will use **RobotC**, which is a language that is used to program the Lego Mindstorms robots that we will use.

GETTING STARTED

UNDERSTANDING THE COMPONENTS OF THE ROBOT

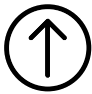
The **Lego Mindstorms EV3** robot that we program consists of several components. The picture below shows all the different components that we will use in the next 3 terms.

Figure 1- Components of the Lego Mindstorms Ev3

Figure 2: The Lego Mindstorms EV3 Forklift

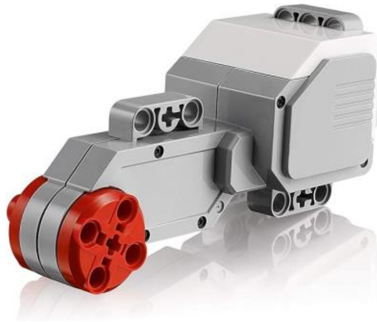
In this section, we will look at some of the components of the robot in a bit more detail.

© 2024 Exceed Robotics ™



THE EV3 BRAIN

The LEGO EV3 Brain is the central control unit that makes everything work. It has 6 buttons and a screen that can be used to navigate and find your programs. The buttons also can be used to interact with your programs as we will find out later. It also has 4 motor ports and 4 sensor ports.



LARGE MOTOR

The *Large Motor* is used for Moving robot wheels. It is continuous which means it does not have any limits; it can keep spinning forever. Our robot will have two large motors: One for the left wheel and one for the right wheel.

MEDIUM MOTOR

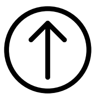
The *Medium Motor* is used to move the arm/forklift up and down. It is not continuous which means it does have limits; it can only turn so far before it stops. Our robot will have one medium motor which is used for controlling the arm.

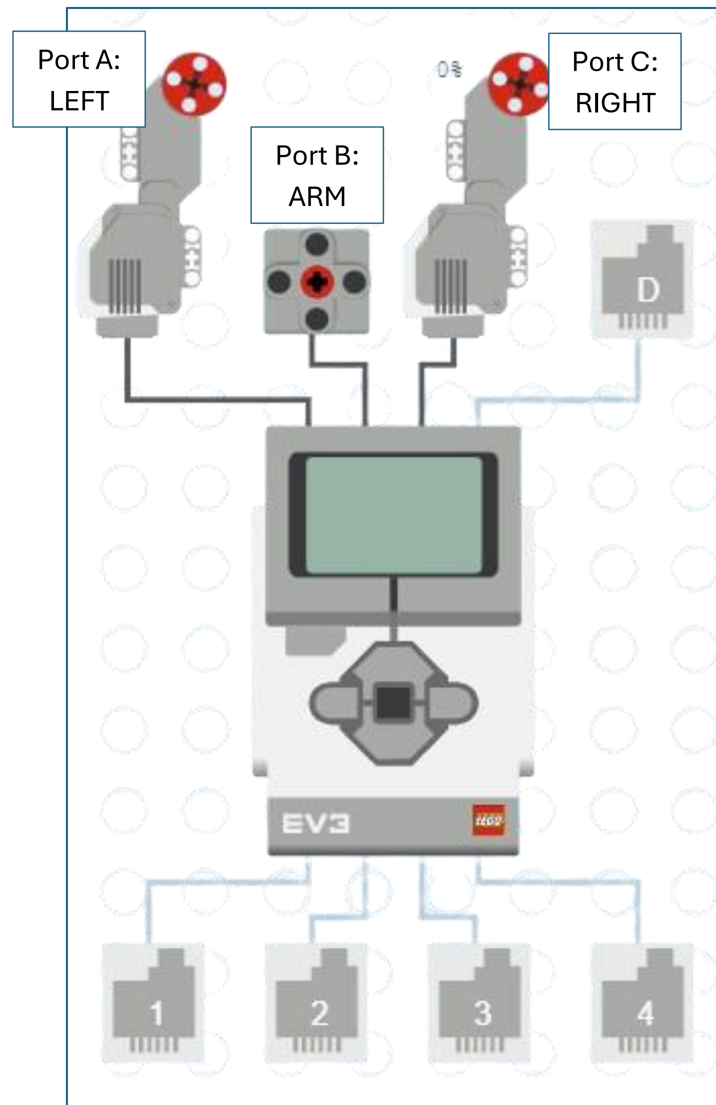
We can also add sensors which the robot uses to understand the environment around it, but they will not be covered until the next term.

ROBOT SETUP FOR THIS COURSE

The robot setup that we are going to use in this course consists of 2 large motors and 1 medium motors. The figure below shows the wiring of the EV3 that we will use for this course.

© 2024 Exceed Robotics ™



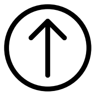


CREATING YOUR FIRST PROGRAM

Now that we have covered the physical parts of the robot. Let's now start programming. As we discussed earlier, we are going to be using RobotC as our programming language. Follow the steps to open RobotC:

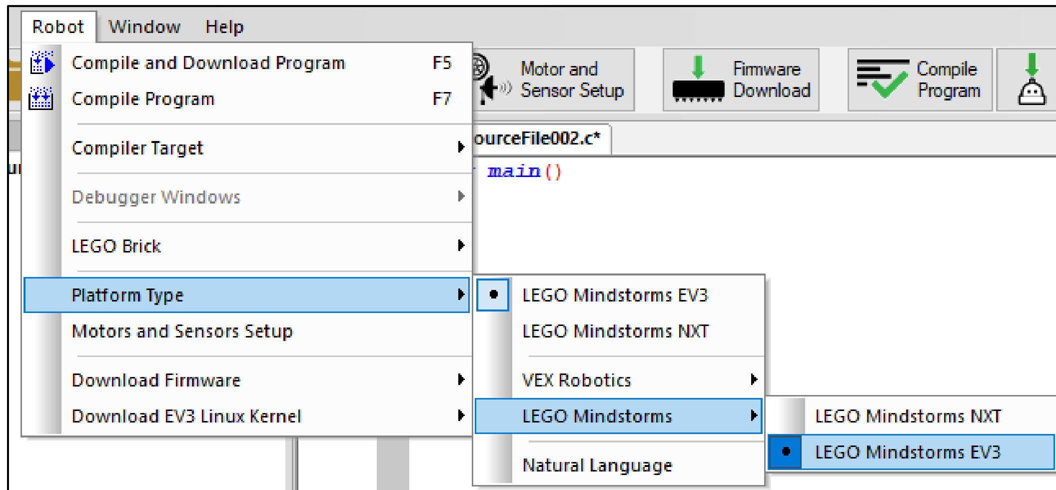
Step 1. To get started, open the RobotC for LEGO Mindstorms application.

© 2024 Exceed Robotics ™

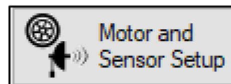


Step 2. Click on New File

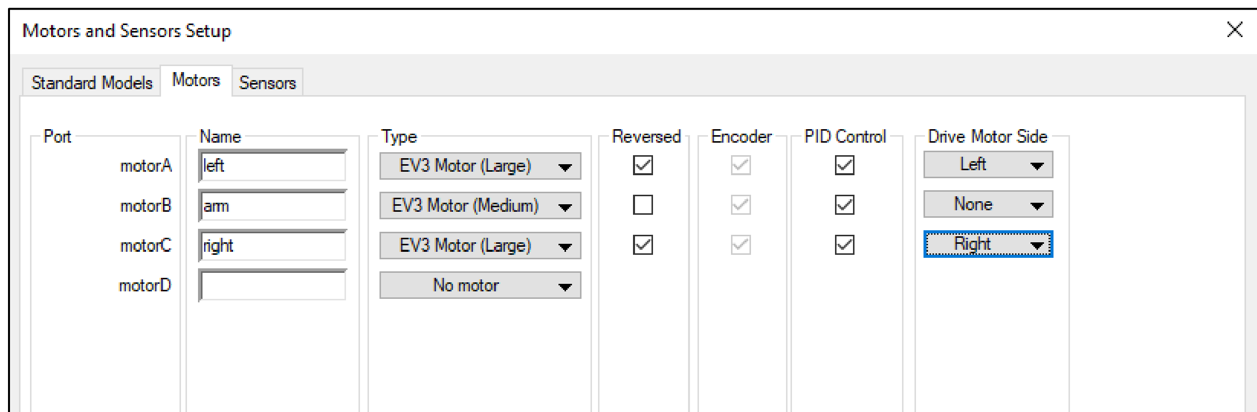
Step 3. Ensure that the platform type is set correctly to Lego Mindstorms EV3



Step 4. Click on Motor & Sensor Setup



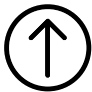
Step 5. In the Motors tab, enter the settings shown below and click Ok



Step 6. On the line just above task main, add the following line of code.

```
#include<exceeders.h>
```


© 2024 Exceed Robotics ™



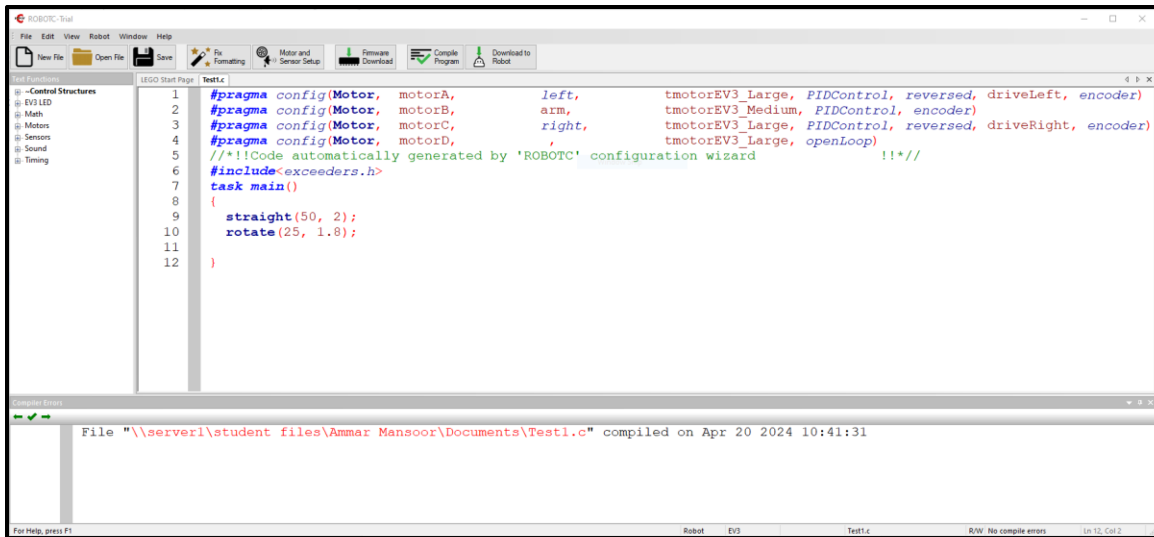
Step 7. **Your code should look like this**

```
Test1.c*
1   #pragma config(Motor,  motorA,          left,          tmotorEV3_Large, PIDControl, reversed, driveLeft, encoder)
2   #pragma config(Motor,  motorB,          arm,           tmotorEV3_Medium, PIDControl, encoder)
3   #pragma config(Motor,  motorC,          right,         tmotorEV3_Large, PIDControl, reversed, driveRight, encoder)
4   #pragma config(Motor,  motorD,           ,             tmotorEV3_Large, openLoop)
5   //*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
6   #include<exceeders.h>
7   task main()
8   {
9
10  }
11
```

Step 8. **Save your work by clicking on the Save icon.**

UNDERSTANDING THE ROBOTC LAYOUT

RobotC for Lego Mindstorms is the application that we will use to write our programs and download our programs to the robot. In this section, let's look at look at what this application looks like.

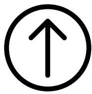


There are three main parts of the layout that we want to look at,

1. Editor

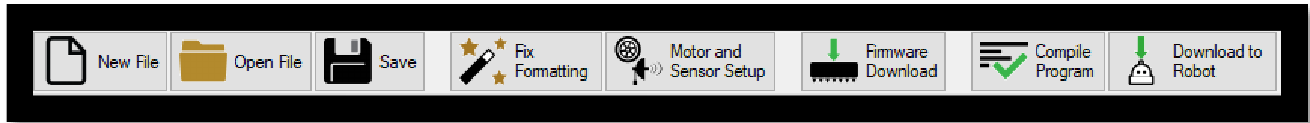
```
LEGO Start Page | Test1.c
1   #pragma config(Motor,  motorA,          left,          tmotorEV3_Large, PIDControl, reversed, driveLeft, encoder)
2   #pragma config(Motor,  motorB,          arm,           tmotorEV3_Medium, PIDControl, encoder)
3   #pragma config(Motor,  motorC,          right,         tmotorEV3_Large, PIDControl, reversed, driveRight, encoder)
4   #pragma config(Motor,  motorD,           ,             tmotorEV3_Large, openLoop)
5   //*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
6   #include<exceeders.h>
7   task main()
8   {
9     straight(50, 2);
10    rotate(25, 1.8);
11
12  }
```


© 2024 Exceed Robotics ™



The Editor is the area where we write our programs.

2. Toolbar



The toolbar consists of tools that we commonly use. Let's take a look at each of these tools.

a) New File: This tool allows you to create a new program file.

b) Open File: This tool allows you to open previously saved files.

c) Save: This tool allows you to save your files in a specific location on your computer.

d) **Fix Formatting**: It makes your code easier to read by adding spaces and tabs to group different blocks such as IF statements, WHILE Loops etc. together (We will look at this tool in action in a later chapter)

Before Fix formatting

```
6   #include<exceeders.h>
7   task main()
8   {
9                                   straight(50, 2);
10      rotate(25, 1.8);
11  straight(50, 2);
12      rotate(25, 1.8);
13                        straight(50, 2);
14  rotate(25, 1.8);
15            straight(50, 2);
16  rotate(25, 1.8);
17
18
19  }
```

After Fix formatting

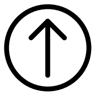
```
6   #include<exceeders.h>
7   task main()
8   {
9     straight(50, 2);
10    rotate(25, 1.8);
11    straight(50, 2);
12    rotate(25, 1.8);
13    straight(50, 2);
14    rotate(25, 1.8);
15    straight(50, 2);
16    rotate(25, 1.8);
17
18
19  }
```

e) **Motor and Sensor Setup:** This allows us to specify what EV3 Devices (motors and sensors) we are using in our current program.

f) Compile Program: Used to check for errors in our program before we download it to the robot. Any errors in the program are highlighted in the console.

g) Download to Robot: Used to download the program to the robot so that we can run our code.

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

3. Console

The console highlights errors in our code. It tells us the kind of error and line number that the error is on.

```
Compiler Errors
    File "\\server1\student files\Ammar Mansoor\Documents\Test1.c" compiled on Apr 20 2024 11:25:20
12  **Error**:Undefined procedure 'rotae'.
12  *Warning*:')' exepected before ';'. Automatically inserted.
13  *Warning*:Missing ';' has been automatically inserted by compiler
15  *Warning*:')' exepected before ';'. Automatically inserted.
For Help, press F1    Robot    EV3    Test1.c    R/W  1 Errors, 3 Warnings    Ln 12, Col 6
```

For example, our code here has *1 Error* *and* *3 Warnings*

An **Error** (Red X) is something that stops our program from running, while a **Warning** (Yellow X) is a mistake that we made that can affect our code. We should work to fix both errors and warnings!

In this case, we have an error on line 12, which you can see by the number *12* is on the left of our Red X – this error says *Undefined procedure 'rotae'*. This means that we made a spelling mistake – we probably meant to write *rotate* instead of *rotae*, and our program is telling us just that.

We also have 3 warnings on lines 12, 13, and 15. The warnings on lines 12 and 15 tell us that the program was expecting a ')' before the ';'

BASIC COMMANDS

DRIVING BASICS

In this section, let's look at the different commands for driving the robot. All these commands require two pieces of information: *Speed* and *Rotations*

1. **SPEED:** how fast the robot will move as a percentage of max speed (-100 to 100)
2. **ROTATIONS:** how many rotations or steps do we want to move; how far (ex. 1, 0.5, 20)

Note: The value you use for speed should always be a whole number. Values such as 30, -20, 100 are all acceptable. However, you cannot use decimal values. On the other hand, the values used for can be any number including decimal values

DRIVING STRAIGHT

To make the robot go straight we can use the straight function. The straight function is written as:

```
straight(speed, rotations);
```

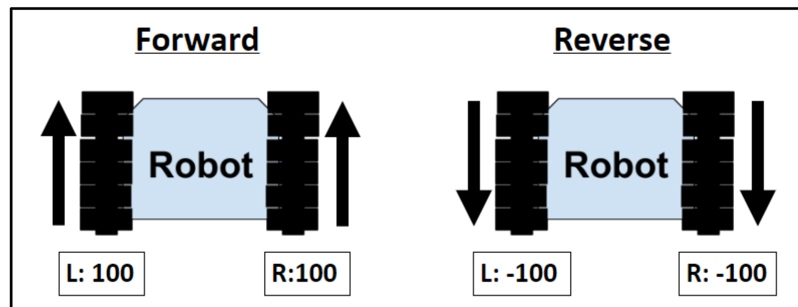

© 2024 Exceed Robotics ™

For example, if want our robot to go forward really fast for 10 rotations we can write the command

straight (100, 10);

The straight command also works for negative values of speed. If the speed is negative the robot goes backwards. For example, if we want our robot to go back 5 rotations slowly we can write **straight (-20, 5);**.

When the **straight();** command is used, both the left and right motors move in the same direction and speed (either both forward or both backwards) as shown.



TURNING

To make the robot turn we use the rotate function. The rotate function is written as shown below,

```
rotate(speed, rotations);
```

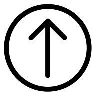
For example, if we want our robot to turn clockwise really fast for 2 rotations we can write the command :

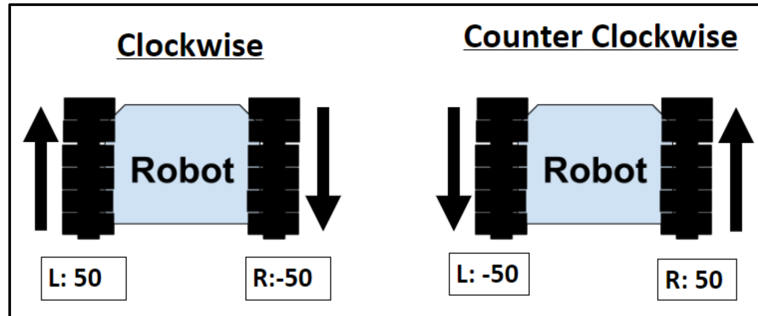
rotate (100, 2);

The rotate command also works for negative values of speed. **If the speed is negative the robot turns in the Counterclockwise direction**. For example, if we want our robot to turn counterclockwise for 5 rotations slowly we can write **rotate (-20, 5);.**

When the **rotate();** command is used, both the left and right motors move in the opposite direction but with the same speed as shown.

© 2024 Exceed Robotics ™





DRIVE COMMAND

The drive command allows you to **control both the left and right motor separately**. This command can give you more control over your robot in certain cases such as making your robot go in a curve.

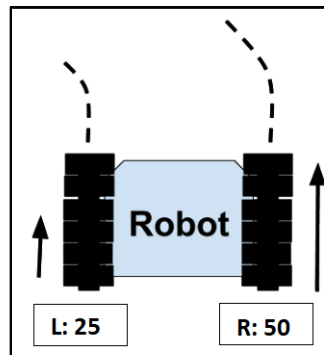
```
drive(LSpeed, RSpeed, Rotations);
```

For example if you want to have your robot curve, you can use the command:

drive(50, 40, 10);

When this command is used the robot's left motor will move at a speed of 50%, it's right motor will move at a speed of 40% and both the motor will turn 10 rotations.

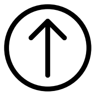
When the **drive();** command is used, the speed and direction of both the left and right motors can be controlled individually. You can have one motor moving at a speed of 50 and the other motor moving at a speed of -25. Just like with the rotate the robot turns in the direction which has the slower speed.



DRIVE TABLE

The drive table below summarizes the number of rotations for your robot's wheels required to move a certain distance or to complete a turn.

 © 2024 Exceed Robotics ™



DRIVE STRAIGHT

Distance (cm)	Motor Rotations
5	1
10	2
20	4
30	6

TURNING (SPIN)

Angle (deg)	Motor Rotations
90	1.8
180	3.6
270	5.4
360	7.2

EXAMPLE: DRIVING BASICS

Task - Write a program to make the robot drive on the path shown.

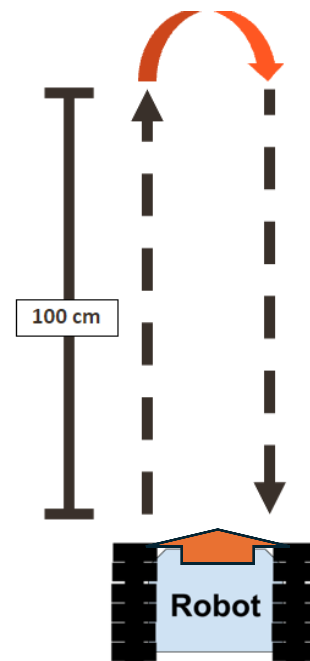
In this example, we are going to make our robot go in the path shown in the image.

We can break this problem into 3 parts. First, we need to go forward 100 cm. Next, we need to rotate clockwise 180 degrees. Finally, we need to go forward 100 cm.

Looking at the tables above we can say that for 100 cm the number of rotations required is 20 rotations. We can then look at the turning table and say that to turn 180 degrees we need 3.6 rotations.

So, we can write the following code:

```
task main ()
{
      straight(50, 20);
      rotate(50, 3.6);
      straight(50, 20);
}
```



USING THE FORKLIFT

In this section, we are going to look at two commands that can be used to control forklifts.

© 2024 Exceed Robotics ™

Unlike the left and right motors, which are continuous motors (meaning they can theoretically rotate forever), the forklift has a fixed range of movement. This means that there are minimum and maximum position limits for the forklift. If you exceed these limits, the robot can break. Before we can move the forklift to a specific position, we need the robot to calculate these limits. We can use the **calibrateLift();** command for this.

```
calibrateLift(motorPort);
```

The ***motorPort*** value is the port on the robot brain where the arm motor is connected (in most cases this will be port B). Therefore, you can use the following **calibrateLift();** command to calibrate the forklift **calibrateLift(motorB);**

Note:- It is extremely good practice to put the **calibrateLift();** command as the first line in our program if we want to use the forklift. Also, the forklift only needs to be calibrated once per program.

Once calibrated, you can use the **liftTo();** command to move the arm motor to a specific position. The **liftTo();** command is written as shown below.

```
liftTo(Speed, Position);
```

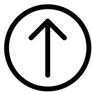
For example, if we want to move the arm up (i.e. 100%) at a speed of 20%, we can write:

```
task main ()
{
	calibrateLift(motorB);
	liftTo(20, 100);
}
```

NOTE: To move the arm down, you don't need to use a negative value of speed in the **liftTo ();** command. The command will automatically move the arm up or down using the new position that we specify. For example, if we want to write a program where the arm goes up all the way (i.e. 100%) and then gets lowered to 50% we can write the following code.

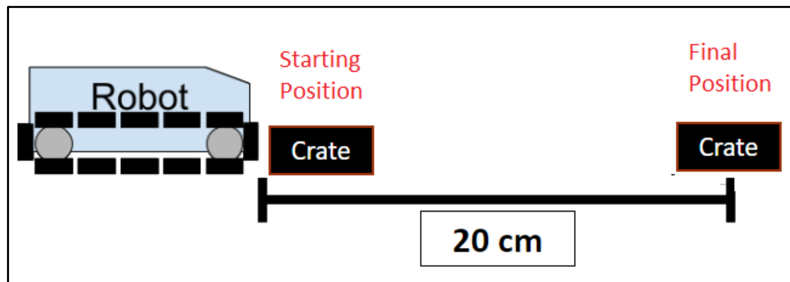
```
task main ()
{
	calibrateLift(motorB);
	liftTo(20, 100);
	liftTo(20, 50);
}
```


© 2024 Exceed Robotics ™



EXAMPLE: USING THE FORKLIFT

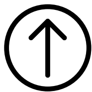
Write a program to make the robot pick up a crate, go forward and then drop the crate on the ground.



Since we are using the forklift in the program, we should first use **calibrateLift();** to calibrate the forklift. Next, let's pick up the crate all the way up (i.e. 100%) slowly at a speed of 20%. Next, we want to go forward 20cm. Looking at the drive table shown above, we can see that for 20 cm we need the robot to go straight for 4 rotations. Finally, we want the arm to go all the way down (i.e. 0%). The code for this program is shown below.

```
task main ()
{
	calibrateLift(motorB);
	liftTo(20, 100);
	straight(20, 4);
	liftTo(20, 0);
}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

SYNTAX – THE RULES OF PROGRAMMING

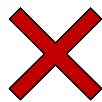
WHAT IS SYNTAX?

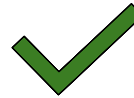
All languages have rules. English, for example, has rules such as a sentence should end with a period, a sentence should start with a capital letter etc. In spoken languages, this set of rules is called grammar. Similarly, there are rules for programming languages. These rules are called **Syntax**.

In this section, we will discuss some of the important pieces of syntax that you should remember for RobotC.

1. All statements (commands) should end with a semi-colon "**;**"

In programming languages like RobotC, a semi-colon tells the robot when a command has ended. It's like putting a period at the end of a sentence in English. If a ";" is not added to our code we will get the following warnings in the console.





```
task main ()
{
        straight (50, 20);
        rotate (50, 3.6);
        straight (50, 20)
}
```

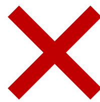
```
task main ()
{
        straight (50, 20);
        rotate (50, 3.6);
        straight (50, 20);
}
```

```
9   *Warning*:Missing ';' has been automatically inserted by compiler
11  *Warning*:';' expected before '}'. Automatically inserted by compiler
```

2. Any bracket that is opened must be closed with a matching bracket "**()**"

There are different types of brackets used in RobotC. **{}** are used to group blocks of code or to define the beginning and end of functions, loops, or conditional statements (we will learn what these are later). **( )** brackets are used when using functions or commands. It's crucial to ensure that every opening bracket has a corresponding closing bracket to avoid syntax errors. For instance, if you start a loop with "**{**", you need to end it with "**}**" to define the scope of the loop.

© 2024 Exceed Robotics ™



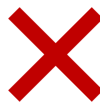
```
task main ()
{
        straight (50, 20);
        rotate 50, 3.6);
        straight (50, 20)
```

```
task main ()
{
        straight (50, 20);
        rotate (50, 3.6);
        straight (50, 20);
}
```

```
7  **Error**:Ummatched left brace '{'
10 **Error**:Expected->'}'. Found 'EOF'
```

3. Make sure your spelling of commands is correct

In programming languages, commands and keywords have specific spellings that must be followed exactly. Misspelling a command can lead to errors in your code. It's essential to pay attention to the spelling of commands and use them accurately.



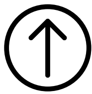
```
task main () {
        straght (50, 20);
        rotae (50, 3.6);
        straight (50, 20)
}
```

```
task main () {
        straight (50, 20);
        rotate (50, 3.6);
        straight (50, 20);
}
```

```
8 **Error**:Undefined procedure 'straiht'.
9 **Error**:Undefined procedure 'roate'.
```

As we continue our journey with **RobotC** we will add a few more items to our list of syntax.

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

EXAMPLE: FINDING ERRORS

Task – Look at the code shown. The code has a few errors. Find them and then rewrite this code.

```
task main (>
{
	calibeteLift(motorB)
	STRAIGHT(50, 10);
	rotate(25,3;
}
```

So there are a few errors here

- In the first line, **the opening round bracket "(" does not have a matching ")" bracket**
- In the second line, there are two errors.
    - **calibrateLift** is spelled incorrectly
    - **there is no ';'** at the end of the statement
- In the third line **straight is written in uppercase**.
- In the fourth line, **there is no closing ")"** before the semi-colon.

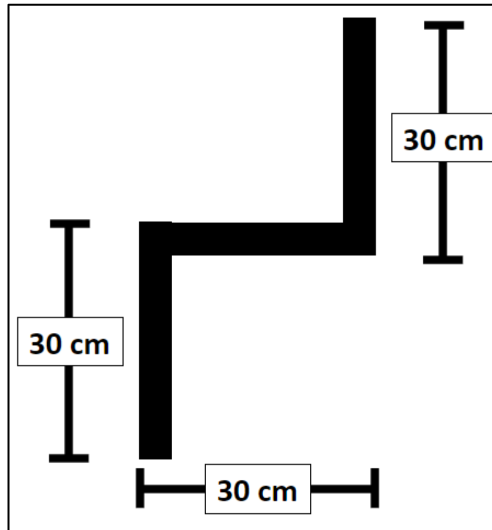
Here is a rewritten correct piece of this code.

```
task main ()
{
	calibrateLift(motorB);
	straight(50, 10);
	rotate(25,3);
}
```


© 2024 Exceed Robotics ™

IN CLASS EXERCISES

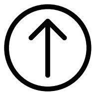
1. Write a program to have the robot drive on the path shown below



Write your code here:

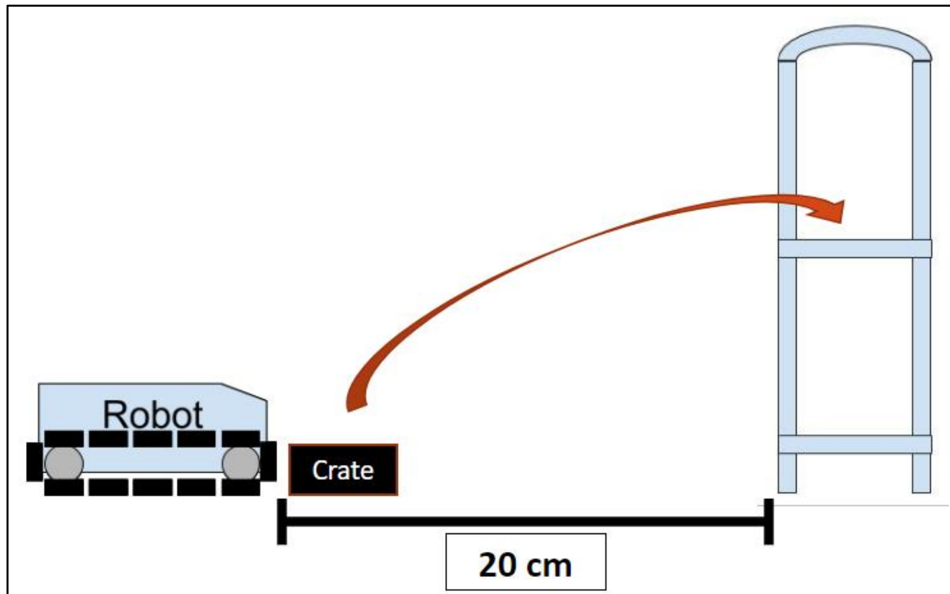
```
task main()
{

}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

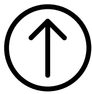
2. **Write a program to have the robot stack the crate on the top shelf.**



Write your code here:

```
task main()
{

}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

HOMEWORK

WHAT IS THE COMMAND?

In this exercise, you must fill in the blanks with the appropriate commands to control the robot's movements and functions. Choose the correct commands to match each.

1. To make the robot go forward at a speed of 25% for 10 rotations.

```
straight(_____, _____);
```

2. To make the robot rotate right at a speed of 20% for 3 rotations.

```
rotate(_____, 3);
```

3. To calibrate the forklift

```
__________(motorB);
```

4. To move the forklift halfway up at a speed of 15%

```
__________(15, _____);
```

5. To make the robot go backwards at a speed of 60% for 20cm (need to convert cm to rotations)

```
__________(-60, _____);
```

6. To make the arm go all the way down at a speed 35%.

```
liftTo(_____, 0);
```

7. To make the robot turn left 90 degrees at a speed of 35.

```
rotate(_____, _____);
```

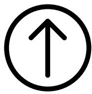
8. To make the robot do a full spin (360 degrees) clockwise at a speed of 50%.

```
rotate(_____, _____);
```

9. To make the robot do half of a spin (180degrees) counterclockwise at a speed of 75%

```
rotate(_____, _____);
```


© 2024 Exceed Robotics ™



WHAT DOES IT ALL MEAN?

Read each line carefully and describe what you think will happen when the robot executes the provided commands. Consider factors like speed, direction, and any specific functions mentioned in the code.

Write down your explanation for each line in the space provided.

```
straight(50,3);
```

```
rotate(-50, 3.6);
```

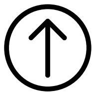
```
calibrateLift(motorB);
```

```
liftTo(25, 100);
```

```
straight(-100, 3.6);
```

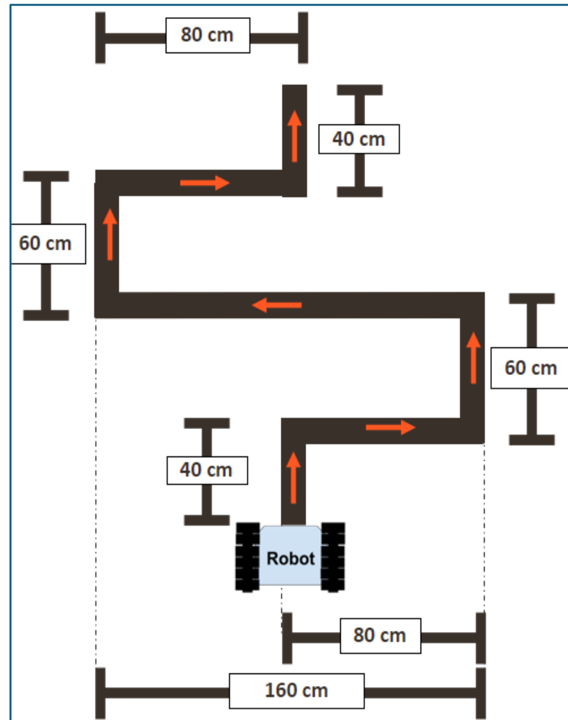
```
rotate(-25,3.6);
```


© 2024 Exceed Robotics ™



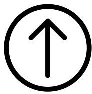
COMPLETE THE CODE

Complete the code so that the robot can drive the path shown:




```
task main ()

{
      straight (100, 8);
      rotate (___ , 1.8);
      straight (____ , 16);

      ____________________
      straight (100, 12);
      rotate (____, 1.8);
      straight (100, 32);
      rotate (______, 1.8);
      straight (100, 12);
      rotate (50, 1.8);
      straight (____ , 16);
      rotate (-50, ______);
      straight(100 , _______);
}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

CONVERTING FROM INSTRUCTIONS TO CODE

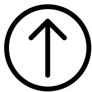
Read carefully the following instructions written in English. Convert them into RobotC code. Look out, some commands give you the rotation value, while others require you to convert from centimetres to rotations.

Instructions in English

1. Start of program
2. Calibrate the lift motor
3. Move forward for 10 rotations really fast
4. Move Arm to 100% at a slow speed
5. Turn right at a speed of 25% and for 90 degrees.
6. Move forward for 10cm slow
7. Move arm down to 50% at a slow speed
8. Move backward for 10cm fast
9. Turn left 90degrees at a speed of 25%
10. Move backward 10rotations fast
11. Move arm all the way to the end
12. End of program

RobotC code goes here

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

WEEK 2 – VARIABLES

WHAT ARE VARIABLES?

In programming, a **variable** can store a value that can be changed. Think of a variable like a box where you can keep numbers or information that might change while you're running your program. By changing what's in the box (the variable), your program can make smart choices based on new information it gets while the robot is moving and doing tasks. In robotics, we often use these variables to remember what the sensors on the robot are seeing. Then, based on what the sensors tell us, the robot can decide what to do next. This helps the robot adapt to different situations and perform tasks better.

Another great thing about variables is that you can use them over and over again for different parts of your program. Once you store information in a variable, you can reuse that same variable in many commands, which makes programming easier and your code cleaner. This means you don't have to keep re-entering the same data; just use the variable wherever you need it!

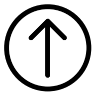
TYPES OF VARIABLES

When you're programming, especially when working with variables, you need to know about data types. **Data types** tell the computer what kind of data you plan to store in a variable and how the computer should use it. There are four main data types in RobotC.

- **int** (Integer) — This data type is used for **whole numbers**, no decimal points
- **float** (Floating Point) — Use this data type for **decimal numbers**
- **bool** (Boolean) — This data type can only hold one of **two states** (True/False, 0/1, on/ off)
- **char** (Character) — This data type is for storing **single characters**

Data Types	Description	Example
int	Integer (whole numbers only)	1, 2 , 0 , -1
float	Real numbers (contain decimal point)	3.14, -0.25, 2.0
bool	Binary	T/F or 1/0
char	Character	‘a’ , ‘!’ , ‘@’

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

EXAMPLE: TYPES OF VARIABLES

What type of variable would you use to store the following pieces of information?

- The **speed** at which a robot is going forward
- The **name** of the robot 'R'
- The **number of rotations** that a robot should turn

To answer this question, we need to think about what values we expect to store in each of these values. We discussed in the previous chapter that the values for speed should always be a whole number and that decimal values are not accepted. Therefore, we should use an int type when creating this variable.

To store the name of the robot inside a variable we need to use a char type variable.

We also discussed in the previous chapter, that the value for rotations can be any type of number including decimals. Hence, we will use the float type for this.

HOW TO CREATE VARIABLES?

To create a variable there are two steps;

Step 1. Declare a variable

To declare a variable, you need to specify the type of variable and the name of the variable.

```
Type VariableName;
```

Step 2. Assign data to a variable

Once we have declared a variable, we can now assign a value to it. To assign data to a variable, we can use the = symbol ("assignment operator").

```
VariableName = Value;
```

EXAMPLE: HOW TO CREATE VARIABLES

Write a program where we create 2 variables, mySpeed which needs to be assigned a value of 50, and myPi which needs to be assigned a value of 3.14.

We can start this program by declaring the variables *mySpeed* and *myPi*. The type of the variable *mySpeed* will be an Integer since the value we are storing is 50, which is a whole number. The type of the variable *myPi* will be a float since the value we are storing is 3.14.

© 2024 Exceed Robotics ™

Next, let's assign the specific values to the two variables. We can use the assignment operator "=" for this. The code for this program is shown below.

```
task main ()
{
        int mySpeed;
        float myPi;
        mySpeed = 50;
        myPi = 3.14;
}
```

VISUALIZING COMPUTER MEMORY

So far, we have looked at how we can declare variables and assign values to variables. Let's now try and understand what happens inside the computer when you create a variable.

Just like the human brain, the computer also stores information in its memory. You can think of the computer memory as a table that contains 3 columns: **Name**, **Type** and **Value**. We will call this table the **Memory Box**. This a powerful technique that you can use when you are troubleshooting (finding errors) in your program.

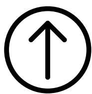
Variable Name	Variable Type	Variable Value
bob	int	25
myRotations	float	3.14

In the above memory box, you can see that we have two variables: an ***int*** variable called bob with a value of 25 and a float variable called ***myRotations*** with a value of 3.14.

1. Let's look, step by step at how this memory box gets filled. Let's look at the code below. The arrow shows you what line of code the computer is now trying to execute.

```
task main ()   <-
{
        int mySpeed;
        float myPi;
        mySpeed = 50;
        myPi = 3.14;
}
```


© 2024 Exceed Robotics ™



As you can the program is at the beginning of the code. At this time the memory box will be empty.

Variable Name	Variable Type	Variable Value

2. Since there is no code to execute, the program will move to the next line of the program.

```
task main ()
{
➡     int mySpeed;
      float myPi;
      mySpeed = 50;
      myPi = 3.14;
}
```

This line tells the computer to create an *integer* variable called ***mySpeed***. This will get added to the Memory Box.

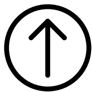
Variable Name	Variable Type	Variable Value
mySpeed	int	

3. Note that ***mySpeed*** still does not have a value attached to it. Since the computer has executed this line let's move to the next line.

```
task main ()
{
      int mySpeed;
➡     float myPi;
      mySpeed = 50;
      myPi = 3.14;
}
```

Like the previous line, this line asks the computer to create a float variable called ***myPi***. Let's add this to our Memory Box.

© 2024 Exceed Robotics ™



Variable Name	Variable Type	Variable Value
mySpeed	int	
myPi	float	

4. Let's move on to the next line.

```
task main ()
{
        int mySpeed;
        float myPi;
        mySpeed = 50;
        myPi = 3.14;
}
```

In this line, we are telling the computer that we want to assign a value of 50 to the variable ***mySpeed***. Let's add this piece of information to our memory box.

Variable Name	Variable Type	Variable Value
mySpeed	int	50
myPi	float	

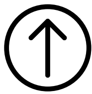
5. Let's move on to the next line.

```
task main ()
{
        int mySpeed;
        float myPi;
        mySpeed = 50;
        myPi = 3.14;
}
```

In this line, we are telling the computer that we want to assign a value of 3.14 to the variable ***myPi***. Let's add this piece of information to our memory box.

Variable Name	Variable Type	Variable Value
mySpeed	int	50
myPi	float	3.14

© 2024 Exceed Robotics ™



6. Let’s move on to the next line.

```
task main ()
{
      int mySpeed;
      float myPi;
      mySpeed = 50;
      myPi = 3.14;
}
```

At this point, **we have reached the end of the program**. There is nothing left to execute.

Using the memory box is a powerful technique that you should use if you are running into any errors in your program.

USING VARIABLES YOU CREATE IN YOUR PROGRAM

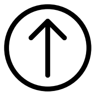
Since we have now created variables. Let’s look at how to use them with the drive commands that we learned such as ***straight();***, ***rotate();***, and ***liftTo();***.

Let’s continue working on the previous piece of code and add some commands to it. The previous code and the memory box looked like this:

```
task main ()
{
      int mySpeed;
      float myPi;
      mySpeed = 50;
      myPi = 3.14;
}
```

Variable Name	Variable Type	Variable Value
mySpeed	int	50
myPi	float	3.14

© 2024 Exceed Robotics ™



7. At the end of our original code, let's **add a straight command.**

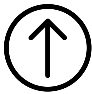
```
task main ()
{
      int mySpeed;
      float myPi;
      mySpeed = 50;
      myPi = 3.14;
  ➡   straight( mySpeed, 2);
}
```

What do you think will happen when the computer tries to execute this line? Does it know what ***mySpeed*** is? Yes, according to the computer memory, the value of ***mySpeed*** is 50. So before running the straight command, the computer will replace the variable name ***mySpeed*** with its value which is 50. This will look something like this:

```
task main ()
{
      int mySpeed;
      float myPi;
      mySpeed = 50;
      myPi = 3.14;
  ➡   straight( 50, 2);
}
```

The computer will now execute the straight command, which will cause the robot to go forward for 2 rotations at a speed of 50%

© 2024 Exceed Robotics ™



8. Let's **add another line** to turn with our robot:

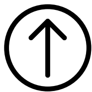
```
task main ()
{
       int mySpeed;
       float myPi;
       mySpeed = 50;
       myPi = 3.14;
       straight(mySpeed, 2);
       rotate( 20, myPi);
}
```

What do you think will happen now? Is there a variable called ***myPi***? Does it have a value? Yes, there is a variable called ***myPi*** and it has a value of 3.14. Before executing this line, the computer will replace the variable name ***myPi*** with its value – which is 3.14. This would look something like this.

```
task main ()
{
       int mySpeed;
       float myPi;
       mySpeed = 50;
       myPi = 3.14;
       straight( mySpeed, 2);
       rotate( 20, 3.14);
}
```

The computer will now execute the rotate command, which will cause the robot to rotate clockwise for 3.14 rotations at a speed of 20%.

© 2024 Exceed Robotics ™



9. Let's try something different else:

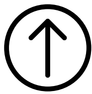
```
task main ()
{
	int mySpeed;
	float myPi;
	mySpeed = 50;
	myPi = 3.14;
	straight( mySpeed, 2);
	rotate( 20, myPi);
	straight(-mySpeed, myPi);
}
```

What do you think will happen now? Does the program know what ***mySpeed*** and ***myPi*** are? What is going to happen to the negative symbol before ***mySpeed***? Well, let's use the same logic that we used before and replace the variable names with their values. What would that look like?

```
task main ()
{
	int mySpeed;
	float myPi;
	mySpeed = 50;
	myPi = 3.14;
	straight( mySpeed, 2);
	rotate( 20, myPi);
	straight(-50, 3.14);
}
```

Aha! This looks like straight with a negative value for speed. This should make the robot go backwards at a speed of 50% for 3.14 rotations.

© 2024 Exceed Robotics ™



10. Let's try something different now. Let's try to **use a variable that does not exist:**

```
task main ()
{
        int mySpeed;
        float myPi;
        mySpeed = 50;
        myPi = 3.14;
        straight( mySpeed, 2);
        rotate( 20, myPi);
        straight(-mySpeed, myPi);
        liftTo(liftingSpeed, 50);
}
```

What do you think will happen now? Does the program know the value of ***liftingSpeed***? No. Since the computer does not know the value of ***liftingSpeed*** it will give an error (see below).

```
15 ✗ **Error**:Undefined variable 'liftingSpeed'. 'short' assumed.
```

At this point, we have reached the end of the program. There is nothing left to execute.

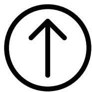
UPDATING YOUR VARIABLES INSIDE YOUR CODE

Variables are very useful because we can change the values stored inside them whenever we want. Let's look at the following example.

```
task main ()
{
        int speed123;
        speed123 = 25;
        straight(speed123, 2);
        speed123 = 50;
        straight(speed123, 4);
}
```

1. In the first line, we are creating a variable called ***speed123***. The type of the variable will be set to int. In the next line, we will assign ***speed123*** with a value of 25. At this point, the memory box looks like this.

© 2024 Exceed Robotics ™



Variable Name	Variable Type	Variable Value
speed123	int	25

2. Let's move on to the next line.
   This line of code will make the robot go **forward at a speed of 25 for 2 rotations**.

```
task main ()
{
        int speed123;
        speed123 = 25;
  ==>   straight(25, 2);
        speed123 = 50;
        straight(speed123, 4);
}
```

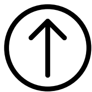
3. Let's move on to the next line:

```
task main ()
{
        int speed123;
        speed123 = 25;
        straight(speed123, 2);
  ==>   speed123 = 50;
        straight(speed123, 4);
}
```

In this line, we are **assigning the value of *speed123* to 50**. This will change the value stored inside the memory box.

Variable Name	Variable Type	Variable Value
speed123	int	50

© 2024 Exceed Robotics ™



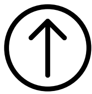
4. Let's move on to the next line:

```
task main ()
{
      int speed123;
      speed123 = 25;
      straight(speed123, 2);
      speed123 = 50;
➡     straight(50, 4);
}
```

Since the value of *speed123* is now 50, this line would make the robot move **forward at a speed of 50 for 4 rotations**.

At this point, we have reached the end of our program. Notice how by changing the value stored inside a variable we were able to make the robot go forward at two different speeds. First at a speed of 25 and then at a speed of 50.

© 2024 Exceed Robotics ™



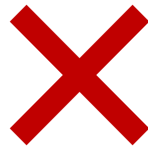
EXCEED ROBOTICS

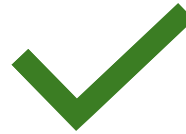
SYNTAX FOR VARIABLES

In the previous chapter, we introduced the idea of **Syntax**. We said that a Syntax is programming rule and discussed four pieces of Syntax. Let's add a few more to it.

1. Variable names can include numbers and letters but no special characters

Variable names are like identifiers used by the computer to store and retrieve data. They can consist of alphanumeric (mix of letters and numbers) characters but should not contain any special symbols such as punctuation marks or emojis. For example, names such as ***myVariable*** and ***playerScore*** are allowed. However, names such as ***my$Variable*** and ***player#Score*** are not allowed.



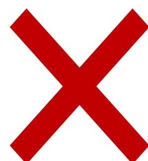


```
task main () {
	int my$Variable;
	float player#Score;
}
```

```
task main () {
	int mySpeed;
	int playerScore;
}
```

2. Variable names must always start with a letter

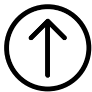
Variable names must begin with a letter of the alphabet. They cannot start with a number. For example, names such as ***x1*** and ***speed123*** are allowed. However, ***1stPlace*** and ***123Bob*** are not allowed.



```
task main ()
{
	int 123speed;
	float 123abc;
}
```

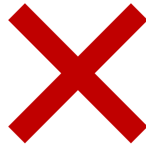
```
task main ()
{
	int speed123;
	int rotations5;
}
```

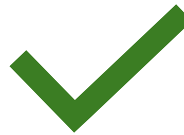

© 2024 Exceed Robotics ™



3. Variable names must not contain spaces (however underscore is allowed)

Variable names should not contain any spaces. While spaces are not permitted, underscores can be used to separate words within a variable name.



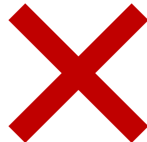


```
task main ()
{
        int drive speed;
        float player score;
}
```

```
task main ()
{
        int driveSpeed;
        int playerScore;
}
```

4. Variable names must be unique (no duplicates, even if they are different types)

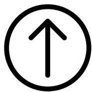
Each variable must have a unique name. This ensures clarity and avoids confusion when we are using variables within a program, regardless of their data types.



```
task main ()
{
        int drive;
        float drive;
}
```

```
task main ()
{
        int driveSpeed;
        float driveRotations;
}
```

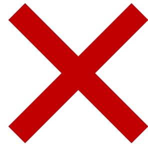

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

5. Variable names should give the programmer hints as to what is stored inside them

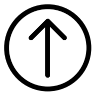
Variables names that describe what is stored inside them, make it easier to read code. They act as labels, helping the programmer understand what is getting stored in the variable and how it is used. For example, if we want to create a variable that stores the number of rotations that the robot moves forward, a good variable name would be ***forwardRotations***.



```
task main ()
{
        float rotations1;
        float rotations2;
}
```

```
task main ()
{
        int driveRotations;
        int turnRotations;
}
```


© 2024 Exceed Robotics ™

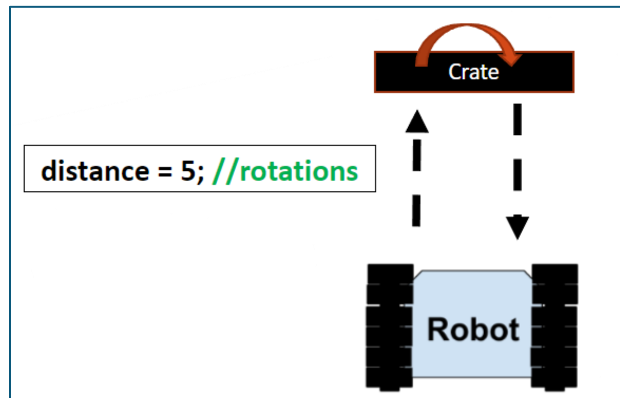


EXCEED ROBOTICS

IN-CLASS EXERCISES

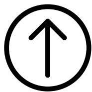
1. Let's dive into a programming adventure with your robot! Your mission is to code your robot to travel back and forth, carrying a crate. The twist? You'll be using a variable named ***mydistance*** to tell your robot how far to go.

   a. Write a program that makes the robot follow the path shown and pick up the crate. Use a variable called *mydistance* to control how far your robot goes forward/backwards.



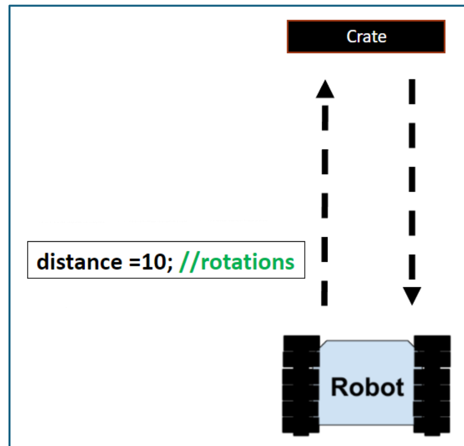

```
task main()
{

}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

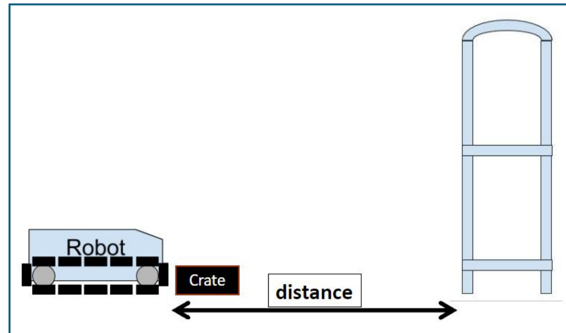
b. **Next, by changing only one line of your program make the robot follow the modified path shown below**




```
task main()
{

}
```

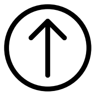

© 2024 Exceed Robotics ™

2. **Write a program to have your robot pick up and place the crate on the top shelf. Use a variable to control how far the robot moves forward/backwards. You can call this variable *distance***




```
task main()
{

}
```

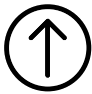

© 2024 Exceed Robotics ™



3. **In the program that you wrote for the previous task add a second variable that controls the speed at which your robot moves forward/backwards**

```
task main()
{

}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

HOMEWORK

CHECK YOUR UNDERSTANDING

Read each code carefully and answer the questions

1. **Code #1**

```
task main ()
{
	int myFwdSpeed;
	bool isItRaining;
	myFwdSpeed = 25;
	int myBwdDistance;
	myBwdDistance = 3.5;
}
```

a. How many variables are created in the code above?

b. What is the type of the variable ***myFwdSpeed***?

c. What is the type of the variable ***isItRaining***?

d. What does the line "**bool isItRaining**" do?

e. Something is wrong with ***myBwdDistance***. Can you figure it out? (Hint: What type should it be?) Fix the issue and enter the correct line of code below.

© 2024 Exceed Robotics ™

2. Code #2

```
task main ()
{
        int a;
        float b;
        int c;
        a = 25;
        b = 3.5;
        c = a + 5;
        straight(a, 5);
        rotate(-c, b);
}
```

a. How many int variables are there in the code above?

b. At what speed will the robot go forward when the code runs?

c. The robot will turn clockwise. Is this statement True or False:

d. How many rotations will the robot turn?

e. What is the turning speed of the robot?

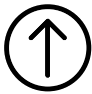
CONVERTING FROM INSTRUCTIONS TO CODE

Read carefully the following instructions written in English. Convert them into RobotC code. Look out, some commands give you the rotation value, while others require you to convert from centimetres to rotations.

Instructions in English

1. Start of program
2. Create an integer variable called **drivingSpeed**

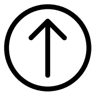
 © 2024 Exceed Robotics ™



3. Create an integer variable called **turningSpeed**
4. Create a float variable called **driveRotation**
5. Create a float variable called **turnRotations**
6. Set the value of **drivingSpeed** to be 100
7. Set the value of **turningSpeed** to be 25
8. Set the value of **driveRotation** to 4.0
9. Set the value of **turnRotation** to the rotations needed for a 90degree turn
10. Make the robot go forward at **drivingSpeed** for **driveRotations**.
11. Make the robot turn 90 degrees using **turningSpeed** and **turnRotation**.
12. End of program

RobotC code goes here

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

FILL IN THE MEMORY BOX

Read each code carefully and fill in the values for how the memory box will look like at the end of the program.

1. **Code #1**

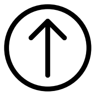
```
task main ()
{
	int myDist;
	float turningRot;
	char myName;

	myDist = 25;
	turningRot = 6.5;
	myName = “A”;
}
```

Variable Name	Variable Type	Variable Value

2. **Code #2**

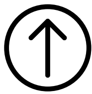
```
task main ()
{
	int num1;
	num1 = 5;
	float pi;
	pi = 3.14;
	char letter;
	letter = 'A';
	num1 = 7;
	pi = 3.14159;
}
```


© 2024 Exceed Robotics ™



Variable Name	Variable Type	Variable Value

© 2024 Exceed Robotics ™



WEEK 3 & 4 – IF STATEMENTS

DISPLAY TEXT ON THE EV3 SCREEN

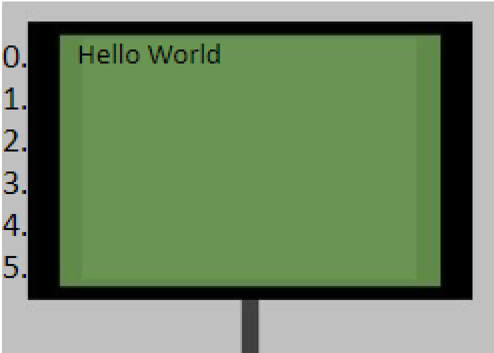
In earlier chapters, we explored the concept of computer memory as an imaginary "memory box." This box organizes and stores the names, values, and types of variables we've created. One problem with this memory box is that it is not visible. It would be incredibly helpful to be able to see the values of these variables right on the EV3 screen. To do this, we can use specific commands to display text.

1. Displaying just text

```
print(lineNumber, text);
```

This command is used to display text on the EV3 screen. You need to specify the ***lineNumber*** and the ***text***. The EV3 screen is organized into six lines, numbered from 0 to 5. The ***lineNumber*** determines which line the text will appear on. When writing the text, make sure to enclose it in double quotes. For instance, to display the “Hello World" on the first line, you can use the following code:

```
task main ()
{
      print(0, “Hello World”);
}
```

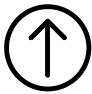


2. Displaying variable value

```
printVar(lineNumber, Variable);
```

This command is designed to display the value of a variable. For example, if you have a variable named ***myAge*** with the value 25, you can show this value on the EV3 screen with ***printVar***. Here's how you might write the code:

© 2024 Exceed Robotics ™



```
task main (){
      int myAge;
      myAge = 25;
      printVar(2, myAge);
}
```

WHAT IS CONDITIONAL PROGRAMMING?

So far, the programs we've written have been straightforward, with the robot following a series of commands in sequence. But what if we need the computer to make decisions based on the situation at hand?

Let's consider an example of a car at an intersection controlled by a traffic light. Depending on the colour of the traffic light, the actions the car should take will vary. If the light is red, the car needs to stop. If it's yellow, the car should slow down, and if it's green, the car can proceed forward. How do we add this "intelligence" to our robots, so that it performs specific actions only under certain conditions? This is what is known as **conditional programming**.

To write a conditional program, we need to first learn Boolean expressions. Boolean expressions are statements that evaluate to either True or False. You can think of Boolean expressions as questions that we are asking the computer. However, these can only be True or False questions. For example, questions like **Is it raining? is the light red?** Etc.

Once we have the Boolean expressions written we can use them with IF statements to write a conditional program. Here is an example of a conditional program.

IF (it is raining) then

Carry an umbrella

IF (it is NOT raining) then

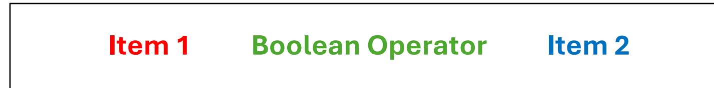
Wear shorts

In the above program, the text shown in **green** is the conditionals and the text shown in **blue** are the actions taken if the Boolean expressions evaluate to **True**. According to the above program If it rains then we should carry an umbrella. On the other hand, if it NOT raining then we should wear shorts.

HOW DO WE WRITE BOOLEAN EXPRESSIONS?

© 2024 Exceed Robotics ™

As we have just learned, Boolean expressions are statements that evaluate to True or False. To write Boolean expressions we follow the following format.



Boolean operators are used for comparison. There are several boolean operators available,

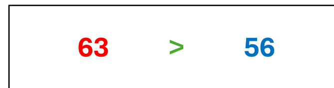
- Equals to (==)
- Less than (<)
- Less than or equal to (<=)
- Not equal to (!=)
- More than (>)
- More than or equal to (>=)

Let's take a look at some examples.

To check if 5 is **equal to** 5 we can write the following

To check if 10 is **not equal** to 25, we can write the following

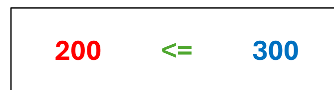
To check if 63 is **greater than** 56, we can write the following



EVALUATING BOOLEAN EXPRESSIONS

As we discussed earlier Boolean expressions return only boolean values, either True or False. So let's look at some conditions and try to figure out what they would return.

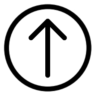
Let's take the following code.



The Boolean expressions is checking whether **200 is less than or equal to 300**. In this case the Boolean expressions will evaluate to True since 200 is less than 300.

Let's try another one.

© 2024 Exceed Robotics ™



The Boolean expressions above is checking whether **71 is more than or equal to 80**. In this case, the Boolean expressions will be evaluated as False, since 71 is less than 80.

Let's try a more complex one.

22 + 30   **>**   **45 -3**

This Boolean expression requires an additional simplification step. Before we check the conditional we want to simplify the terms on the right and left side of the boolean operator. We can replace the expression on the Left-Hand side with 52 because 22 + 30 gives you 52. We can replace the expression on the Right-Hand side with 42 because 45 – 3 gives you 42. Let's make these changes.

52   **>**   **42**

Ah, much better. We can evaluate this much more easily. The conditional is checking whether **52 is more than 42**. In this case the conditional will result in True.

BOOLEAN EXPRESSIONS WITH VARIABLES

When programming we need to use Boolean expressions that check the values of the variables. Let's take a look at an example of this. Here is a piece of code written in RobotC.

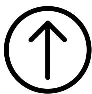
```
task main() {
      int mySpeed;
      mySpeed = 25;
      bool myExpression;
      myExpression = mySpeed < 50;
}
```

In the first two lines, we create a ***int*** variable called ***mySpeed*** and set its value to 25. Let's add this information to our memory box.

Variable Name	Variable Type	Variable Value
mySpeed	int	25

Let's look at the next line.

© 2024 Exceed Robotics ™




```
task main (){
        int mySpeed;
        mySpeed = 25;
  ➡     bool myExpression;
        myExpression = mySpeed < 50;
}
```

In this line, we create a Boolean variable called ***myExpression***. Let's add this to our memory box.

Variable Name	Variable Type	Variable Value
mySpeed	int	25
myExpression	bool	

Let's move to the next line.

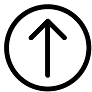
```
task main (){
        int mySpeed;
        mySpeed = 25;
        bool myExpression;
  ➡     myExpression = mySpeed < 50;
}
```

Just like we did before when working with variables, we want to start by replacing the variable names with their values. This will look something like this.

```
task main (){
        int mySpeed;
        mySpeed = 25;
        bool myExpression;
  ➡     myExpression = 25 < 50;
}
```

Now let's evaluate the boolean expression: is 25 less than 50. Since 25 is less than 50, this conditional evaluates to **True**. We can replace the conditional expression with its value. This would look something like this.

© 2024 Exceed Robotics ™



```
task main (){
      int mySpeed;
      mySpeed = 25;
      bool myExpression;
      myExpression = true;
}
```

Let's also update the memory box after the previous statement.

Variable Name	Variable Type	Variable Value
mySpeed	int	25
myExpression	bool	true

At this point, we have reached the end of our program so our program will exit.

```
task main (){
      int mySpeed;
      mySpeed = 25;
      bool myExpression;
      myExpression = true;
}
```

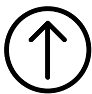
WHAT IS AN IF STATEMENT?

An "**IF statement**" in programming is a powerful tool that enables you to make decisions based on conditions. It allows your robot to perform different actions depending on whether certain conditions are met or not. By using IF statements, you can add intelligence to your robot, making it respond to different events.

THE STRUCTURE OF AN IF STATEMENT

An **IF** statement has two parts: A ***conditional*** and the ***actions***.

© 2024 Exceed Robotics ™



```
if (myVariable < 10)
{
        straight(50, 2);
}
```


The conditional consists of one or more Boolean expressions (we can do more than one, as we will see later). In the actions section, we specify the different actions we want to complete if the Boolean expression evaluates to True. The above IF statement checks if the value stored inside ***myVariable*** is less than 10. If true, the robot will go straight at a speed of 50% for 2 rotations.

HOW DOES AN IF STATEMENT WORK (EXAMPLE 1)

In this section, we want to take a look at how an IF statement works. Take a look at the code below and let's see how it works.

```
task main(){
      int myNumber;
      myNumber = 12;
      if (myNumber >= 10){
            print(2, "Woah! Big Number");
      }
      straight(100, 2);
}
```

1. In the lines highlighted we create an int variable called myNumber and store the number 12. Let's add this to the memory box and then move onto to the next line.

Variable Name	Variable Type	Variable Value
myNumber	int	12

© 2024 Exceed Robotics ™

```
task main(){
      int myNumber;
      myNumber = 12;
➡     if (myNumber >= 10){
            print(2, "Woah! Big Number");
      }
      straight(100, 2);
}
```

2. We are at an IF statement. We can start by evaluating the boolean expression. To simplify this expression, let's replace the variable ***myNumber*** with its value. From our memory box we know that the variable ***myNumber*** has a value of *12*.

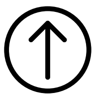
```
task main(){
      int myNumber;
      myNumber = 12;
➡     if (12 >= 10){
            print(2, "Woah! Big Number");
      }
      straight(100, 2);
}
```

3. Now let's evaluate the boolean expression. The expression is checking whether 12 is greater than or equal to (>=) 10. This expression will be True, since 12 is greater than or equal to 10. Since the expression is True, the program will move inside the actions part of the IF statement.

```
task main(){
      int myNumber;
      myNumber = 12;
      if (myNumber >= 10)
➡     {
            print(2, "Woah! Big Number");
      }
      straight(100, 2);
}
```

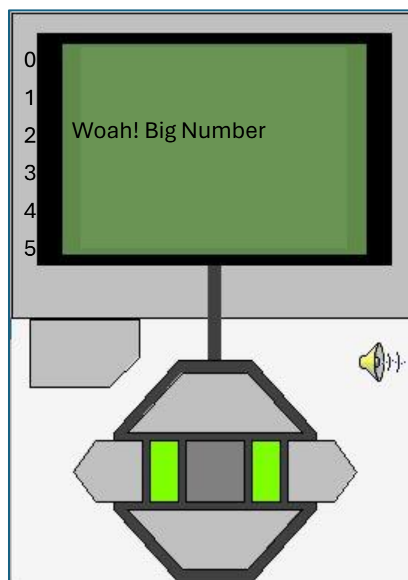
4. As you can see the pointer has moved inside the actions. The program will perform each of the actions specified inside the actions. Let's move on to the next line.

© 2024 Exceed Robotics ™



```
task main(){
        int myNumber;
        myNumber = 12;
        if (myNumber >= 10)
        {
            print(2, "Woah! Big Number");
        }
        straight(100, 2);
}
```

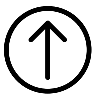
5. In this line, we are printing the text "***Woah! Big Number***" on line 2 of EV3 screen. Let's move to the next line.




```
task main(){
        int myNumber;
        myNumber = 12;
        if (myNumber >= 10)
        {
            print(2, "Woah! Big Number");
        }
        straight(100, 2);
}
```

6. On this line, we have performed all the actions inside the IF statement. The program will move outside the IF statement. Let's move to the next line.

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

```
task main(){
	int myNumber;
	myNumber = 12;
	if (myNumber >= 10)
	{
		print(2, "Woah! Big Number");
	}
	straight(100, 2);
}
```

7. This line would cause the robot to go forward really fast for 2 rotations. Let's move on to the next line.

```
task main(){
	int myNumber;
	myNumber = 12;
	if (myNumber >= 10)
	{
		print(2, "Woah! Big Number");
	}
	straight(100, 2);
}
```

8. At this point we have reached the end of our program.

Let's summarize what happened. The IF statement checked if the value of ***myNumber*** is greater than or equal to 10. Since this was True, the robot performed the actions inside the IF statements. In this example, we only had 1 action inside the IF statement which displays the text "Woah! Big Number" on line 2 of the EV3 screen. Next the program exists the IF statement and performs any code present outside of the IF statement. In this case there is one command after the IF statement. This command moves the robot forward at a speed of 100% for 2 rotations.

HOW DOES AN IF STATEMENT WORK (EXAMPLE 2)

Let's look at an example where the conditional part of the IF statement is False. We will work on the same code as the previous example but make one change. Let's walkthrough what happens.

© 2024 Exceed Robotics ™

```
task main(){
        int myNumber;
        myNumber = 5;
        if (myNumber >= 10){
                print(2, "Woah! Big Number");
        }
        straight(100, 2);
}
```

1. Instead of assigning the variable a value of 12 let's change it to 5. Here is what the code and the memory box would look like. Let's move on to the next line

Variable Name	Variable Type	Variable Value
myNumber	int	**5**

```
task main(){
        int myNumber;
        myNumber = 5;
        if (myNumber >= 10){
                print(2, "Woah! Big Number");
        }
        straight(100, 2);
}
```

2. We are at an IF statement. We can start by evaluating the boolean expression. To simplify this expression, let's replace the variable ***myNumber*** with its value. From our memory box we know that the variable ***myNumber*** has a value of 5.

```
task main(){
        int myNumber;
        myNumber = 5;
        if (5 >= 10){
                print(2, "Woah! Big Number");
        }
        straight(100, 2);
}
```


© 2024 Exceed Robotics ™

3. Now let's evaluate the boolean expression. The expression is checking whether 5 is greater than or equal to 10. This expression will be False, since 5 is NOT greater than or equal to 10. Since the expression is False, the program will move to the end of the IF statement.

```
task main(){
      int myNumber;
      myNumber = 5;
      if (myNumber >= 10)
      {
            print(2, "Woah! Big Number");
  ➡   }
      straight(100, 2);
}
```

4. At this point we have reached the end of the IF statement. Let's move to the next line.

```
task main(){
      int myNumber;
      myNumber = 5;
      if (myNumber >= 10)
      {
            print(2, "Woah! Big Number");
      }
  ➡   straight(100, 2);
}
```

5. In this line, the robot will move forward at 100% speed for 2 rotations. Let's move on to the next line.

EXAMPLE: WRITING IF STATEMENTS

Task: Add an IF statement to the program shown, which checks if the value stored inside the variable *myNumber* is less than 20. IF the variable is less than or equal to 20, the program should print "It is a small number" on line 2 of the EV3 screen.

```
task main(){
      int myNumber;
      myNumber = 12;
}
```


© 2024 Exceed Robotics ™

Let's start by looking at the conditional. The Boolean expression to check if ***myNumber*** is less than or equal to 20 can be written as – ***myNumber <= 20***. Next, we can add the actions that we want to perform. In this task, we want to display the text "It is a small number" on line 2 of the Ev3 screen. We can use the ***print*();** command for this.

Here is the final code for this task.

```
task main(){
        int myNumber;
        myNumber = 12;
        if (myNumber <= 20){
                print(2, "It is a small number");
        }
}
```

USING THE EV3 BUTTONS

The Lego Mindstorms EV3 brain has a total of 5 buttons (actually 6 but the back button doesn't count). We can use the EV3 buttons to interact with the programs that we write. One of the commands used to interact with the EV3 robot is shown below.

```
waitForButton();
```

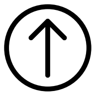
The ***waitForButton();*** command can be used to get the button values from the robot. This value can be then stored inside a variable.

So what does the ***waitForButton();*** command do? The command does 3 things. First, it stops the program. The robot will stop, and no other code will be executed in the meantime. Next, the buttons on the robot Ev3's brain will start blinking red. This indicates that the program is waiting for us to press one of the buttons. Finally, when we press the button, the command returns the value assigned to that button. Let's look at an example of using the Ev3 buttons.

Let's look at the following code.

```
task main (){
        int myButton;
        myButton = waitForButton();
}
```


© 2024 Exceed Robotics ™



At the start of our program, the memory box is empty.

Variable Name	Variable Type	Variable Value

Let's move to the next line.

```
task main (){
  ➡  int myButton;
     myButton = waitForButton();
}
```

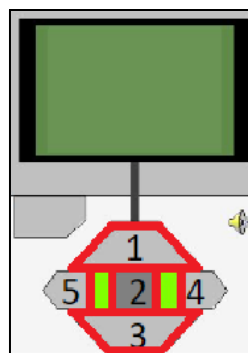
In this line, we declare an integer variable called ***myButton***. Let's look at what this does to the memory box.

Variable Name	Variable Type	Variable Value
myButton	int	

Let's move to the next line.

```
task main (){
     int myButton;
  ➡  myButton = waitForButton();
}
```

In this line, the computer sees the ***waitForButton()*** command. Let's recall what we said the command does. First, it stops the program and the robot. Next, it makes the Ev3 lights blink red. Finally, it returns the number associated with the button that we pressed.



© 2024 Exceed Robotics ™

Let's say that we press the middle button. This button has a value of 2. The command then returns this value. This means that the entire ***waitForButton();*** command can be replaced by the number 2.

```
task main (){
        int myButton;
  ➡     myButton = 2;
}
```

Now what do you think the code does? It assigns the value of 2 to the variable ***myButton***.

Variable Name	Variable Type	Variable Value
myButton	int	2

We can use the buttons to interact with the program and perform different sets of actions when different buttons are pressed.

EXAMPLE USING A BUTTON AND IF STATEMENTS.

Task: Write a program that uses the Ev3 buttons and IF statements. IF button 1 is pressed the robot should move forward one rotation. IF button 2 is pressed the robot should move backward one rotation.

We start the program by creating a variable to store the button values that we get. We can call this variable ***myButton*** and make this of type *int*.

```
task main(){
        int myButton;
}
```

We can now add the ***waitForButton();*** command now.

```
task main(){
        int myButton;
        myButton = waitForButton();
}
```

Now that we have the variable setup and the program reading from the Ev3 brain we can write our IF statements.

© 2024 Exceed Robotics ™

Our 1st IF statement needs to check if button 1 is pressed and make the robot move forward if this is the case. The conditional for this IF statement can be written as "myButton == 1". The action for this IF statement just contains the straight command. This IF statement can be written as shown.

```
task main(){
	int myButton;
	myButton = waitForButton();
	if (myButton == 1){
		straight(50, 1);
	}
}
```

Similarly, we need to add another IF statement, which checks if button 2 is pressed and makes the robot go backwards.

```
task main(){
	int myButton;
	myButton = waitForButton();
	if (myButton == 1){
		straight(50, 1);
	}

	if (myButton == 2){
		straight(-50, 1);
	}
}
```

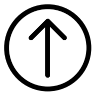
SYNTAX FOR IF STATEMENTS

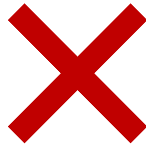
Let's add a few more Syntax rules to keep in mind for IF statements.

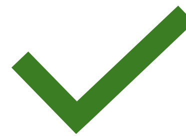
1. Enclose Conditions in Parentheses

Make sure that the condition in your IF statement is surrounded by parentheses, like this: (and). This tells the computer exactly what to check before deciding what to do next.

© 2024 Exceed Robotics ™





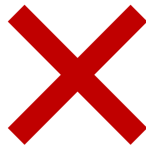


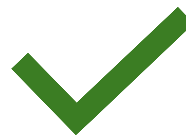
```
task main ()
{
   if (5 < 6
   {
      straight(50,1);
   }
}
```

```
task main ()
{
   if (5 < 6)
   {
      straight(50,1);
   }
}
```

2. Wrap Actions in Braces

Always put the actions that should happen if the condition is true inside braces, { and }. This helps you keep your actions neatly together, making it easier to see what belongs inside the IF statement.





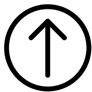
```
task main ()
{
   if (5 < 6)

      straight(50,1);
   }
}
```

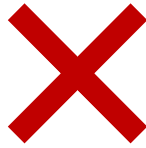
```
task main ()
{
   if (5 < 6)
   {
      straight(50,1);
   }
}
```

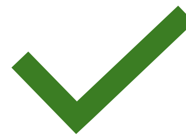
3. No Semicolons on the IF Line

Remember, do not put a semicolon ; right after the IF condition. Most lines in coding need a semicolon, but the IF statement is different. If you put a semicolon right after the IF condition, the computer gets confused and doesn't do what you want.

© 2024 Exceed Robotics ™



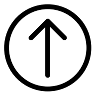




```
task main ();
{
   if (5 < 6)

      straight(50,1);
   }
}
```

```
task main ()
{
   if (5 < 6)
   {
      straight(50,1);
   }
}
```

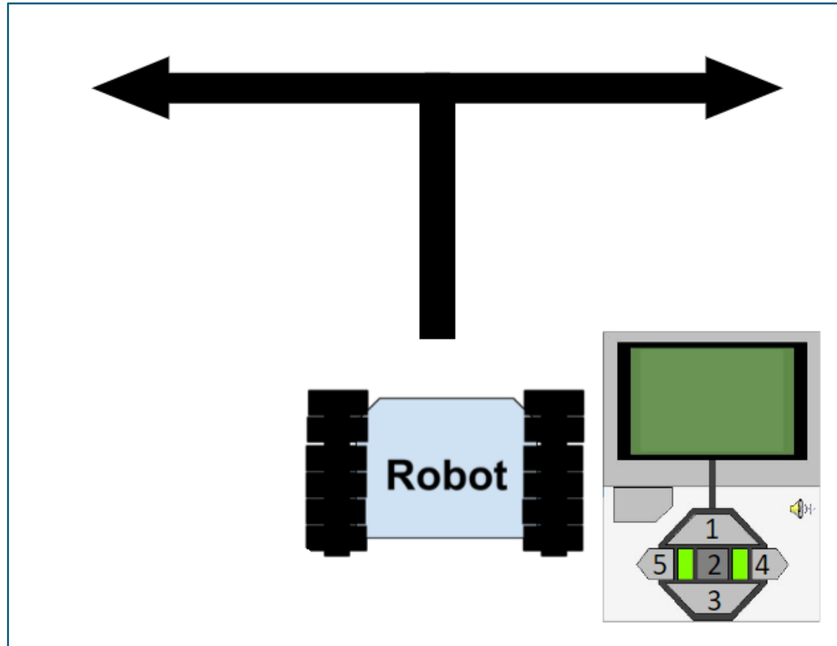

© 2024 Exceed Robotics ™



IN CLASS PROBLEMS

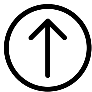
1. Write a program to drive forward, then wait until a button is pressed.
    a. If the left button is pressed, turn left
    b. If the right button is pressed turn right.

   Afterward, continue driving straight




```
task main()
{

}
```

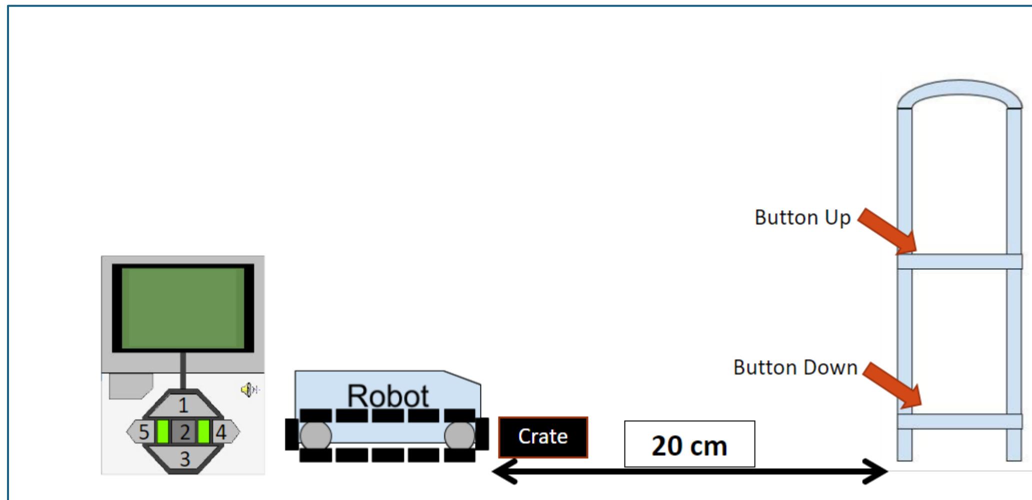
 © 2024 Exceed Robotics ™



EXCEED ROBOTICS

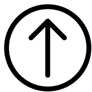
2. Write a program to choose what shelf to place the crate on
   a. If the Up button is pressed, Place the crate on the top shelf
   b. If the Down button is pressed, Place the crate on the bottom shelf

   Additionally, your program must include a variable to control speed!

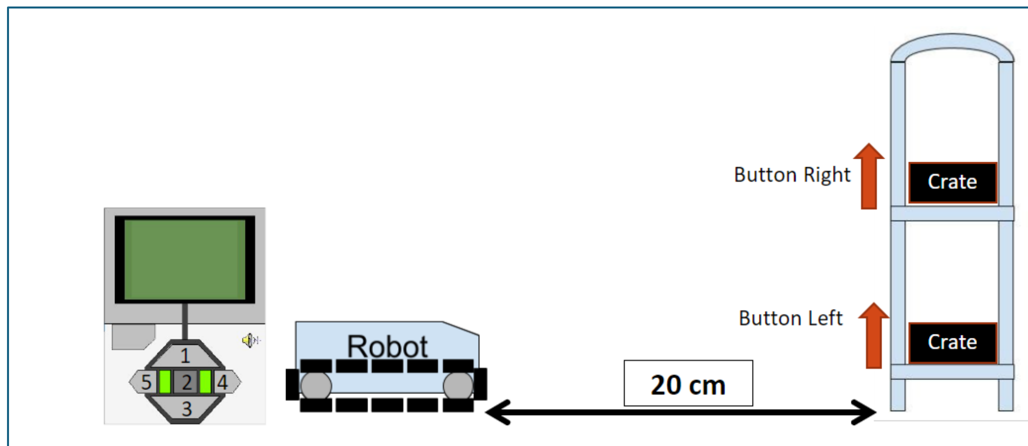



```
task main()
{



}
```


© 2024 Exceed Robotics ™

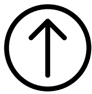


3. Include the left and right buttons to allow the robot to pick up the crate from either shelf
a. If the right button is pressed, pick up the crate on the top shelf
b. If the left button is pressed, pick up the crate on the bottom shelf




```
task main()
{

}
```


© 2024 Exceed Robotics ™



HOMEWORK

CHECK YOUR UNDERSTANDING

1. What are the three actions that take place when the robot reaches a line that says the following?

```
myButton = waitForButton();
```

2. What does the following line do?

```
print( 2, “Hello There!”);
```

3. What does the following line do?

```
printVar( 2, mySpeed);
```

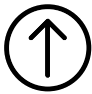
4. What does the expression below check?

```
25 > 30
```

5. Look at the code below. What will happen when the code runs?

```
int myNumber;
myNumber = 90;
if (myNumber == 90)
{
      print(1, “The number is 90”);
}
if (myNumber < 100)
{
      print(2, “The number is small”);
}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

6. Is the value of the Boolean Variable **check** True or False?

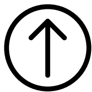
```
int myNumber;
myNumber = 90;
bool check;
check = myNumber < 100;
```

CODING EXERCISES

1. Add two IF statements to the code below
    a. First one should check if the userNumber is equal to secretNumber. If it is True, we should print “You guessed the correct number”
    b. The second IF statement should check if userNumber is NOT equal to secretNumber. If this is the case, we should print “Wrong number! Try again.”

```
task main()
{
      int userNumber;
      int secretNumber;
      secretNumber = 20;
      userNumber = 20;



}
```


© 2024 Exceed Robotics ™

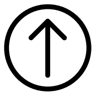


EXCEED ROBOTICS

2. Write a program that first gets which button is pressed.
    a. If Button is 1 – The robot should go forward 1 rotation
    b. If Button is 3 – The robot should go backward 1 rotation
    c. If Button is 4 – The robot turns right 90 degrees
    d. If Button is 5 – The robot turns left 90 degrees

```
task main()
{

}
```


© 2024 Exceed Robotics ™



WEEK 5 – WHILE LOOPS

WHAT ARE WHILE LOOPS?

In programming, a ***WHILE*** loop is a type of loop that runs while a certain condition is met.

A ***WHILE*** loop in programming keeps running as long as a certain condition is true. It's like a repeating IF statement.

Building on what we learned about IF statements from the previous chapter, where we saw how a program can decide based on whether a condition is true or not:

A ***WHILE*** loop relates closely to an ***IF*** statement in that it repeatedly checks if a condition is true. With an ***IF*** statement, if the condition is true, the robot performs the action just once. But with a ***WHILE*** loop, if the condition is true, the robot will perform the action, then recheck the condition, and if it's still true, it will perform the action again. This keeps repeating continues until the condition becomes false. The ***WHILE*** loop can run many times, whereas an ***IF*** statement only runs once for each time it's encountered in the program's flow.

STRUCTURE OF WHILE LOOPS

The structure of a ***WHILE*** loop is similar to an ***IF*** statement. It has two parts conditional and actions.

The conditional consists of one or more Boolean expressions (we can do more than one as we will see later). In the actions, we specify the different actions we want to complete if the Boolean expression evaluates to True. The above WHILE loop checks if the value stored inside ***myVariable*** is less than 10. While this is true, the robot will keep going straight at a speed of 50% and for 2 rotations.

```
while (myVariable < 10)
{
        straight(50, 2);
}
```

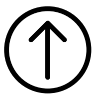
Conditional

Actions

HOW DO WHILE LOOPS WORK?

To understand how ***WHILE*** loops work, let's look at an example.

© 2024 Exceed Robotics ™



```
task main(){
➡ int myNumber;
➡ myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

The first line declares an int variable called ***myNumber***. In the next line, the variable ***myNumber*** is assigned a value of 0. Let's add this to the memory box.

Variable Name	Variable Type	Variable Value
myNumber	int	0

Let's move on to the next line.

```
task main(){
      int myNumber;
      myNumber = 0;
➡     while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

In this line, we have the conditional part of our while loop. The Boolean expression is checking whether ***myNumber*** is less than 3. Is this True or False? It is True because **0 is less than 3**. Let's replace the expression with True.

© 2024 Exceed Robotics ™

```
task main(){
      int myNumber;
      myNumber = 0;
➡     while (true){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

Since the conditional is True, the computer will perform the actions inside the loop.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
➡           straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

The first action inside this ***WHILE*** loop is a straight command. This would make the robot go forward at a speed of 25% for 2 rotations. Let's move to the next line.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
➡           myNumber = myNumber + 1;
      }
}
```

Let's try and understand what this line does. Let's try and simplify this expression by replacing the variable ***myNumber*** on the left-hand side with its value. This would look something like this.

© 2024 Exceed Robotics ™

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = 0 + 1;
      }
}
```

We can simplify this expression since we know that 0 + 1 gives you 1.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = 1;
      }
}
```

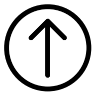
What do you think the program will do now? It will assign the variable ***myNumber*** a value of 1. Let's add this to our memory box.

Variable Name	Variable Type	Variable Value
myNumber	int	1

Let's look at the next line now.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

We have reached the end of the ***WHILE*** loop. Now because this is a ***WHILE*** loop, we must go back and check if our conditional is still True.

```
task main(){
    int myNumber;
    myNumber = 0;
    while (myNumber < 3){
        straight(25,2);
        myNumber = myNumber + 1;
    }
}
```

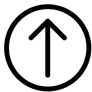
Let's check if our conditional is True or False. Our conditional is checking if the value of the variable ***myNumber*** is less than 3. This is True since the value of ***myNumber*** is 1 and 1 is less than 3.

```
task main(){
    int myNumber;
    myNumber = 0;
    while (true){
        straight(25,2);
        myNumber = myNumber + 1;
    }
}
```

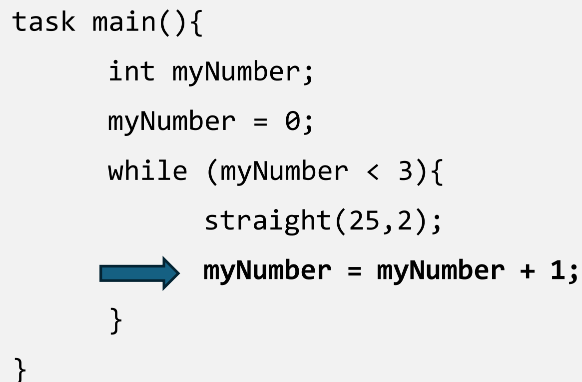
Since the conditional is True. The ***WHILE*** loop will perform the actions inside of it. Let's look at the first action.

```
task main(){
    int myNumber;
    myNumber = 0;
    while (myNumber < 3){
        straight(25,2);
        myNumber = myNumber + 1;
    }
}
```


© 2024 Exceed Robotics ™

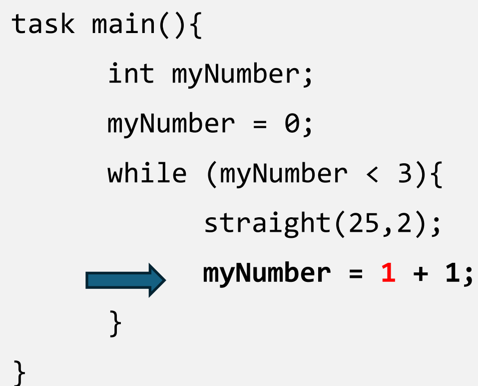


The first action inside this ***WHILE*** loop is a straight command. This would again make the robot go forward at a speed of 25% for 2 rotations. Let's move to the next line.



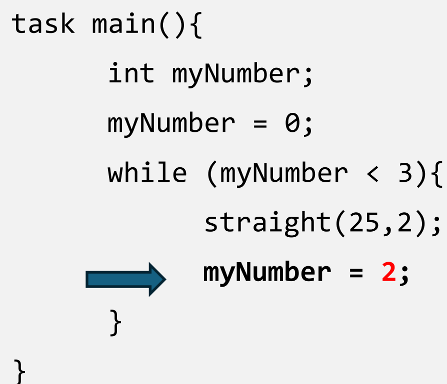
```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

Let's try and simplify this expression by replacing the variable ***myNumber*** on the left-hand side with its value. This would look something like this.



```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = 1 + 1;
      }
}
```

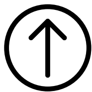
We can simplify this expression since we know that 1 + 1 gives you 2.



```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = 2;
      }
}
```

What do you think the program will do now? It will assign the variable ***myNumber*** a value of 2. Let's add this to our memory box.

© 2024 Exceed Robotics ™



Variable Name	Variable Type	Variable Value
myNumber	int	2

Let's look at the next line now. We have reached the end of the ***WHILE*** loop. Now because this is a ***WHILE*** loop, we must go back again and check if our conditional is still True.

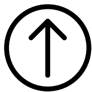
```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

Let's check if our conditional is True or False. Our conditional is checking if the value of the variable ***myNumber*** is less than 3. This is True since the value of ***myNumber*** is 2 and 2 is less than 3.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (true){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

Since the conditional is True. The ***WHILE*** loop will perform the actions inside of it. Let's look at the first action.

© 2024 Exceed Robotics ™



EXCEED ROBOTICS

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

The first action inside this ***WHILE*** loop is a straight command. This would again make the robot go forward at a speed of 25% for 2 rotations. Let's move to the next line.

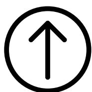
```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

Let's try and simplify this expression by replacing the variable ***myNumber*** on the left-hand side with its value. This would look something like this.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = 2 + 1;
      }
}
```

We can simplify this expression since we know that 2 + 1 gives you 3.

© 2024 Exceed Robotics ™



```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = 3;
      }
}
```

What do you think the program will do now? It will assign the variable ***myNumber*** a value of 3. Let's add this to our memory box.

Variable Name	Variable Type	Variable Value
myNumber	int	3

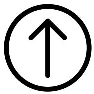
We have reached the end of the ***WHILE*** loop. Now because this is a ***WHILE*** loop, we have to go back and check if our conditional is still True.

```
task main(){
      int myNumber;
      myNumber = 0;
      while (myNumber < 3){
            straight(25,2);
            myNumber = myNumber + 1;
      }
}
```

Let's check if our conditional is True or False. Our conditional is checking if the value of the variable ***myNumber*** is less than 3. This is False since the value of ***myNumber*** is 3 and 3 is NOT less than 3.

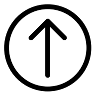
action. Since the conditional is False, the computer exits the ***WHILE*** loop. How many rotations in total did our robot go forward? The robot has gone forward a total of 6 rotations.

© 2024 Exceed Robotics ™



```
task main(){
	int myNumber;
	myNumber = 0;
	while (myNumber < 3){
		straight(25,2);
		myNumber = myNumber + 1;
	}
}
```


© 2024 Exceed Robotics ™

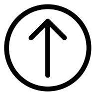


IN-CLASS EXERCISES

1. Write a program to have the robot use a counter to drive 4 (linear) laps.
   A lap consists of
   - Drive Forward 10 rotations
   - Turn 180 degrees turn
   - Drive Forward 10 rotations
   - Turn 180 degrees turn

```
task main()
{

}
```


© 2024 Exceed Robotics ™

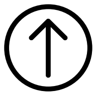


2. Modify the above program to use the EV3 buttons to specify how many laps the robot performs

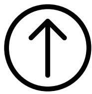
Button	Laps
UP	1
Down	2
Right	3
Left	4

```
task main()
{



}
```


© 2024 Exceed Robotics ™



© 2024 Exceed Robotics ™



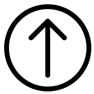
HOMEWORK

CODING EXERCISES

1. Write a program that prints all the even numbers between 0 – 100, on the EV3 screen.

```
task main()
{

}
```


© 2024 Exceed Robotics ™



2. Write a program that make the robot go in a square for a specific number of times.

```
task main()
{

}
```

CHECK YOUR UNDERSTANDING

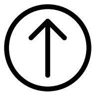
1. What are the three actions that take place when the robot reaches a line that says the following?

```
myVal += 1;
```

2. How many times will the robot go forward in the code below?

```
int myNumber;
myNumber = 90;
while(myNumber < 0)
{
     straight(50, 2);
}
```


© 2024 Exceed Robotics ™



EXCEED ROBOTICS

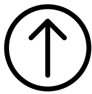
3. There are two errors in the code below. Can you spot them?

```
int numLaps;
numLaps = 0;
while(numLaps < 5
{
     straight(50, 2);

```

4. When the code below is run on the robot the robot goes forward forever? What is missing and where?

```
int numLaps;
numLaps = 0;
while(numLaps < 5)
{
     straight(50, 2);
}
```


© 2024 Exceed Robotics ™

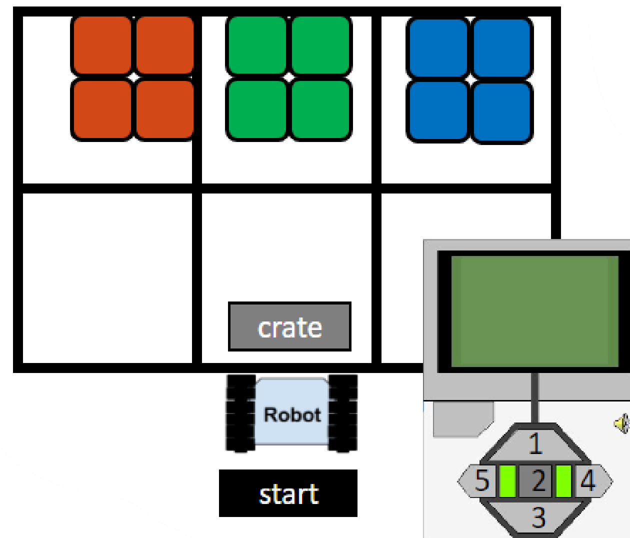


WEEK 6, 7 AND 8 - PROJECT

INTRODUCTION TO THE PROJECT

BACKGROUND

Your robot has been employed by a busy shipping company. This company has several different trucks that need to be loaded. Your robot's job will be to aid the workers in their daily work.



TASK 1 – BASIC PROGRAM

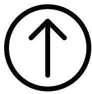
The first task for your robot will be to load any of the trucks (selected by the user).

Your program should use the Ev3 buttons to select which truck the robot will load. Once a truck is loaded the robot should return to where it started.

For example, IF Button 1 is pressed, the crate should be dropped on the middle truck. On the other hand, IF Button 5 is pressed, the crate should be picked and dropped on the truck on left side.

Additionally, depending on how busy the day will be, the workers need to be able to easily edit the speed the robot drives. (This means the worker should be able to edit 1 line of code only and affect the entire program!)

© 2024 Exceed Robotics ™



TASK 2 – TIME-SAVER!

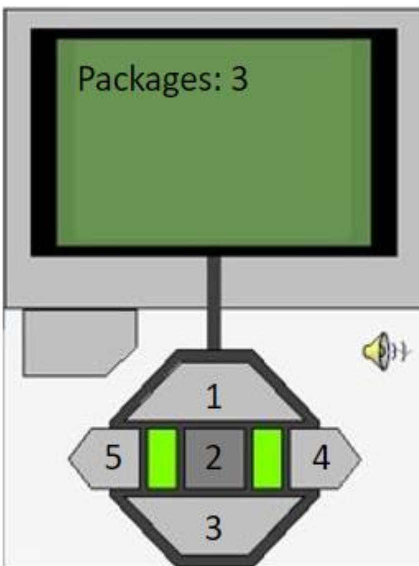
To improve the performance of the robot (and save time), the workers would like the program to continue running once it returns. This should allow the user to easily select another to be load by simply pushing a new button.

Psst. Think of how can I make some actions repeat forever?

TASK 3 – CRATE COUNTERS

During the busy season, the company ships many packages per day, it is difficult for the workers to keep track of how many have been shipped.

Expand your program to keep track of how many crates have been loaded, display the current total on the LCD screen.



© 2024 Exceed Robotics ™