



touch**develop**

Create apps anywhere.
Keyboard optional.

Microsoft®
Research

INDEX

1. Brief Introduction	1
2. Introduction to TouchDevelop.....	3
3. Fundamental Materials of Computer Science.....	9
3.1. Data Types.....	9
3.1.1. Numbers	9
3.1.2. Strings	10
3.1.3. Booleans.....	10
3.1.4. Not Operator	11
3.1.5. And Operator	11
3.1.6. Or Operator	11
3.2. Variables	12
3.3. Invalid Item.....	13
3.4. User Input.....	15
3.5. Operators	19
3.6. If and Else If.....	24
3.7. Loops	26
3.7.1. For Loop	26
3.7.2. While Loop	27
4. Meet the Engduino!	28
4.1. Introduction to Engduino.....	28
4.2. Components in Engduino	28
4.3. Implementation of while and for loops.....	30
4.4. The Button in Engduino	33
4.5. The LEDs in Engduino	34
4.6. The Accelerometer in Engduino	36
4.7. Summary Game (The World Cup Flags Engduino).....	42
5. Final Project (The Engduino Race)	46
5.1. Getting Started	46
5.2. Let's Start Coding	49
5.3. Completing the Pedometer	56
5.4. Challenge Problem	57
6. Teacher Notes.....	58
6.1. Uploading Engduino to TouchDevelop	58
6.2. Bugs with Engduino and TouchDevelop	61

BRIEF INTRODUCTION

In these tutorials we're going to introduce you to the very first step on how to use Microsoft's touchdevelop online coding. In addition, you will also learn some basic fundamentals on how to think like a programmer through some exercises. We will also include some coding knowledge that is commonly used in many programming language, so you can get used to the idea of programming itself. So what are we waiting for? Let's get started!

PART I - INTRODUCING TOUCH DEVELOP

TouchDevelop is an interactive environment for learning programming. It is mainly designed for mobile devices. It lets you create programs that works on mobile platforms such as iOS, android and windows. Go to this link <https://www.touchdevelop.com/>. Create an account and log in. You will be directed to the main page as shown in Figure 1. Select your coding skill by clicking the Skill level tab in the main page

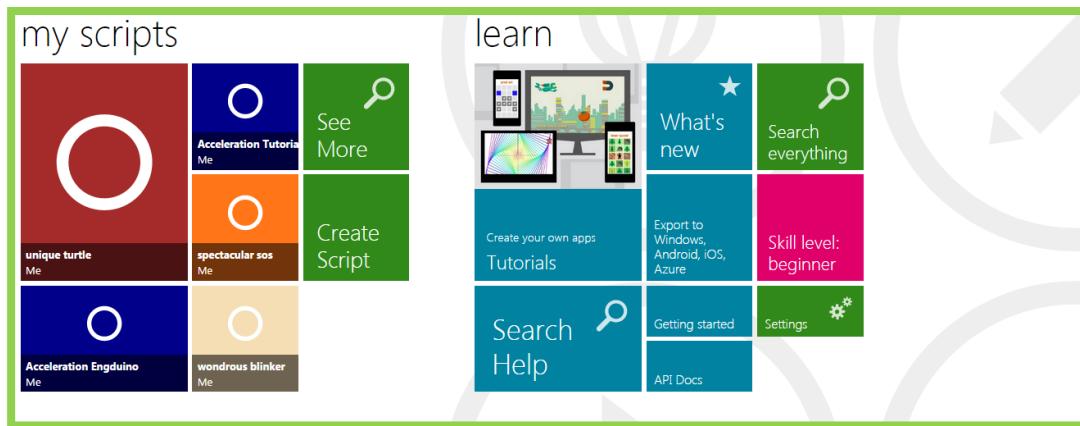


Figure 1 Touchdevelop home

HOW TO PRACTICE WITH EXISTING TUTORIALS

In the main page, click Tutorials. You will be directed to this page shown below. Click the “first steps with turtle” tutorial. If it is not in the featured page you can search for it in the search box in the top left. Also try the other tutorials such as jumping bird, coding jetpack jumper to get yourself familiar with TouchDevelop. If you need more help on using TouchDevelop, check out the TouchDevelop Programming on the Go. Get the book here: <https://www.touchdevelop.com/docs/book>.

the hub

Search here...

featured

- games**
Resources on creating games.
- getting started**
A few tips on getting started with TouchDevelop.
- language**
Language syntax.
- social coding**
Understanding the script lifecycle in TouchDevelop.
- teach**
Teaching with TouchDevelop.
- touchdevelop local**

Coding Jetpack Jumper!

Get introduced to coding by fixing a broken game, then share it with your friends!

by TouchDevelop

jumping bird

A tutorial to build a flappy bird clone.

by TouchDevelop

pixel art

Learn to create your own pixel art!

by TouchDevelop

© 2014 Microsoft. [privacy and cookies](#) [legal](#)

my scripts **run**

```
function main()
    turtle → forward(200)
end function
```

tap there
select that line

SUPER-COOL TURTLE

tutorial: first steps with turtle

© 2014 Microsoft. [privacy and cookies](#) [legal](#)

This is the script editor. To run the program, click the run button. Clicking the + button will add a new line. In this tutorial, the turtle goes forward by 200 meters. You can make the turtle go in different directions and also change the colour.

SOCIAL

The screenshot shows the Microsoft Office 365 Home page. On the left, there's a "social" section featuring a photo of the Eiffel Tower and links for Forums, Facebook, Twitter, and Create Group. On the right, there's a "showcase" section displaying various Office 365 apps like DataFit, eMotions in Motion, 3D Dice Roll, zigzag puzzle, and Block Hop.



the forums

general issues everything

comments

 Hello everyone my name is Joery (Captain Planet) i'm not so familiar with coding outside of TD and hope to someday ... [Captain Planet](#)
13 minutes ago on Zion games

 #pullRequest /yoagcebw /mjkruyoo
... [peli@touchdevelop](#)
13 minutes ago on stcode tutorial

 adding missing (stauto) #publicationNotes
... [peli@touchdevelop](#)
13 minutes ago on stcode tutorial

 I think he meant the news vine
... [Zion Games developer, Jared Gaitan](#)
17 minutes ago on Zion games

 Meant Swedish

 [Captain Planet](#) on [Zion games](#)
Hello everyone my name is Joery (Captain Planet) i'm not so familiar with coding outside of TD and hope to someday publish apps to the windows store, i look forward to meeting the groups community! ;D
13 minutes ago :: eghswfah

[Reply...](#)

 [peli@touchdevelop](#) on [stcode tutorial](#)
#pullRequest /yoagcebw /mjkruyoo
13 minutes ago :: qwwwnwspo

 adding missing (stauto) #publicationNotes

This is the form page, in this page you can ask questions and find solutions to your problems. In the social page you can create groups and share your achievements in Facebook or twitter. So you and your friends can edit each other's code and most importantly brainstorming!

The screenshot shows the TouchDevelop library interface. On the left, there's a sidebar with 'categories' and two main sections: 'games' and 'libraries'. Under 'games', there are links to 'tools' (1208), 'entertainment' (1633), 'education' (967), and 'productivity' (629). Under 'libraries', there are links to 'music' (743) and 'business' (312). To the right of the sidebar is a 'top & new' section featuring a grid of small game screenshots. Each screenshot includes the script name, creator, and some statistics. There are also 'See More' and 'New Scripts' buttons.

This section has the libraries where you see the programs created by other people. You can find a range of functions in these libraries such as datatypes.

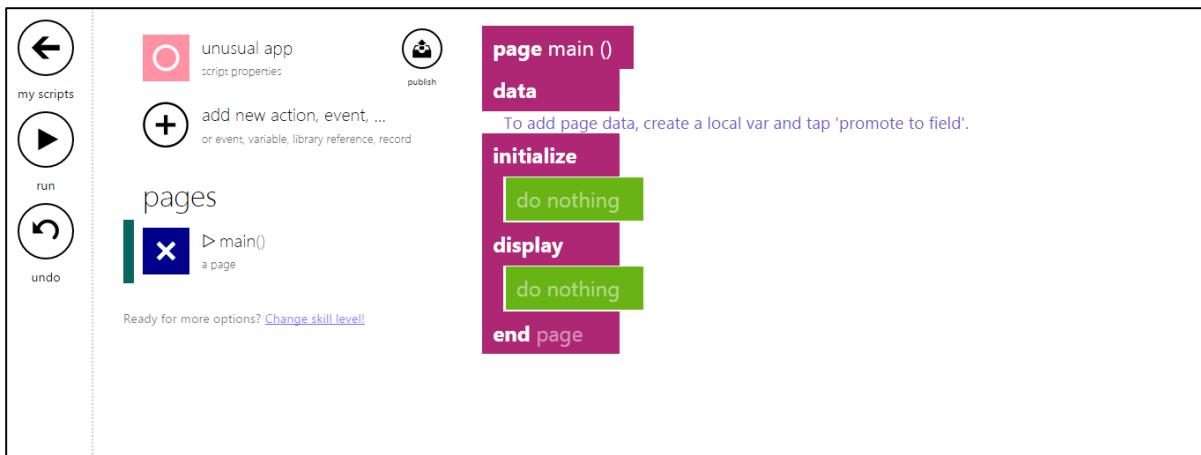
CREATING AN APP

To create an app, click create script, then choose a blank template.

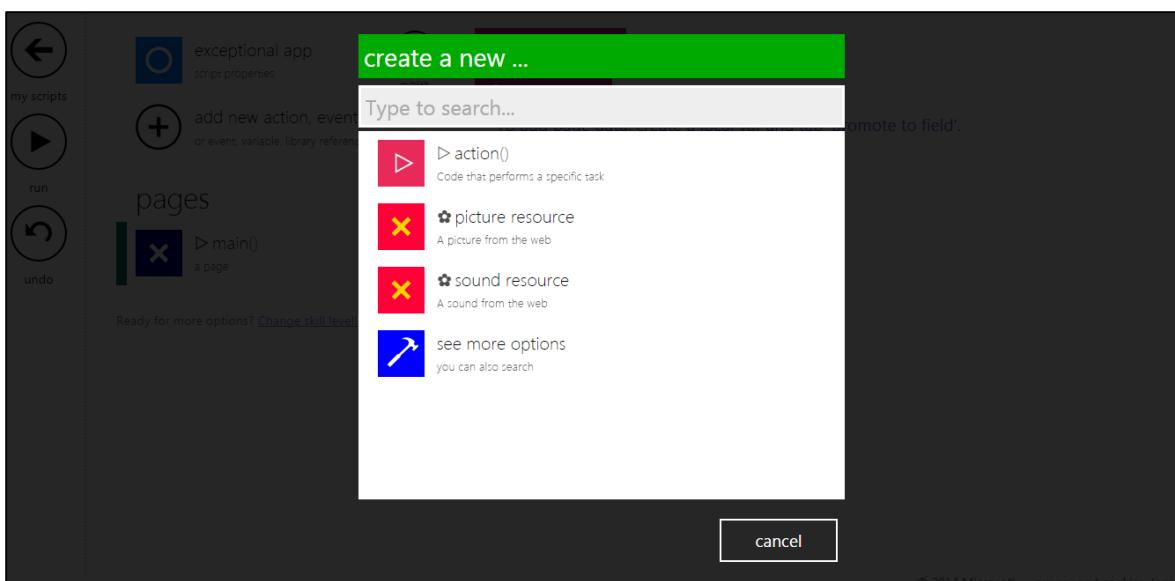
The screenshot shows the 'pick a script template...' dialog. It has a search bar at the top labeled 'search templates'. Below it is a list of template options with icons and descriptions. The 'beginners' category is highlighted in green. The options are:

- blank**: An empty script, which doesn't do anything.
- blank game**: Boiler plate code to create a game.
- blank app**: An empty app using pages and boxes.
- blank turtle**

At the bottom right of the dialog is a 'cancel' button.



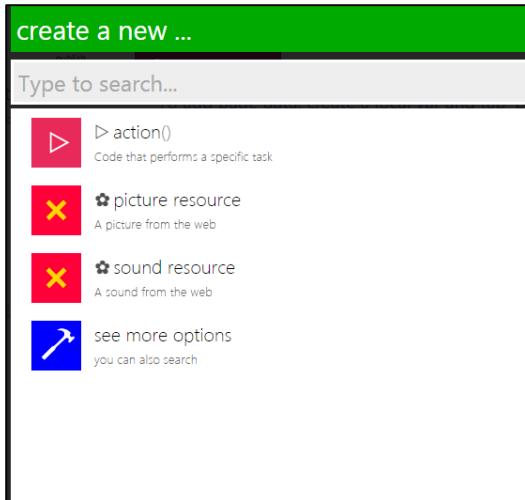
This is an example of blank app template. To create a new action click the plus button and click action, then a function will be created



Well, that's pretty much it to help you start with touchdevelop! But just to make sure it sticks in your brain, we'll just go straight to some tutorials.

CREATING AN ACTION

An action does a task. To create an action, click add new action, the click action. An action contains a function which can be linked to the main function. To link the action to the main function you have to add a line and type the name of the action in the search box and click the action.



exceptional app
script properties

publish

+ add new action, event, ...
or event, variable, library reference, record

function do stuff ()

do nothing

end function

code

▷ do stuff()
an action

This screenshot shows a Scratch-like programming environment. On the left, there are several icons: a blue circle labeled "exceptional app" and "script properties", a "publish" button, a plus sign labeled "add new action, event, ... or event, variable, library reference, record", and a green vertical bar labeled "code". In the center, there is a function definition: "function do stuff ()". Inside the function, there is a single block: "do nothing". Below the function, there is an "end function" block. To the right of the function, there is a green vertical bar labeled "code". At the bottom left, there is a red triangle icon with a white arrow labeled "▷ do stuff()" and "an action".

function main ()

*template:empty}

add

+ templatename:ADJ app

▷ main

+ function

add

This screenshot shows a Scratch-like programming environment. It features a function definition: "function main ()". Inside the function, there is a grey box labeled "*template:empty}" with an "add" button. Below the function, there is a green box labeled "+ templatename:ADJ app" with a plus sign icon. At the bottom, there is a green horizontal bar labeled "▷ main" with a plus sign icon. At the very bottom, there is another green box labeled "+ function" with a plus sign icon.

PART II - FUNDAMENTAL MATERIALS OF COMPUTER SCIENCE

DATA TYPES (NUMBERS, STRING, BOOLEAN)

Numbers

A number can be of any value that is positive, negative or in decimals.

1

☞ we have a Number here; it doesn't do anything by itself; use 'post to wall' to display it



2.1

☞ we have a Number here; it doesn't do anything by itself; use 'post to wall' to display it



In the example above, we have a positive number, 1, and a positive decimal number, 2.1

-5

☞ we have a Number here; it doesn't do anything by itself; use 'post to wall' to display it



-3.9

☞ we have a Number here; it doesn't do anything by itself; use 'post to wall' to display it



In the example above, we have a negative number, -5, and a negative decimal number, -3.9

action +(other : Number) **returns** Number

Adds numbers Example: $3+3 = 9$ (This is the return value)

action /(other : Number) **returns** Number

Divides numbers Example: $10/5 = 2$ (This is the return value)

action =(other : Number) **returns** Boolean

Compares numbers for equality Example: $4 = 4 \rightarrow \text{True}$ (This is the return boolean)

action \geq (other : Number) **returns** Boolean

Compares numbers for more or equal Example : $4 \geq 5 \rightarrow \text{False}$ (This is the return boolean)

action $>$ (other : Number) **returns** Boolean

Compares numbers for more Example : $10 > 2 \rightarrow \text{True}$ (This is the return boolean)

In the example above, we can perform different action with numbers such as addition, subtraction, division or comparison.

STRINGS

Strings are pieces of text within the " "

var s := "this is a string"



s := "hello " || "world" Output : hello world



var count := s → count

Count will have a value of : 16 (spaces are included!)

var first char := s → at(0)

first char will have a value of "t"



In the example above,

- We have a string text of "this is a string" that is saved in a variable called 's'.
- We can also concatenate (add) two or more strings together using the " || " operator
- We can also count the length of the string using the "count" action. (Note that spaces are included in the count value too!)

BOOLEANS (TRUE OR FALSE)

Booleans are "True" or "False"

var t := **true**

var f := **false**



In the example above, we declared a variable and assign the boolean value of "True" to 't' and "False" to 'f'

NOT OPERATOR

- **not true ==> false**
- **not false ==> true**

We can convert a boolean from "True" to "False" using the "NOT" operator and vice versa!

AND OPERATOR

- **true and false ==> false**
- **false and true ==> false**
- **false and false ==> false**
- **true and true ==> true**

In the example above, we use the "AND" operator which takes two boolean and return a boolean.

AS A RULE OF THUMB, IF ANY OF THE BOOLEAN HAS A "FALSE", THE RESULT WILL AUTOMATICALLY BE "FALSE", THE RESULT WILL ONLY BE "TRUE" WHEN THE TWO BOOLEANS ARE "TRUE".

OR operator

- **true or false ==> true**
- **false or false ==> false**
- **true or true ==> true**

IN THE EXAMPLE ABOVE, WE USE THE "OR" OPERATOR WHICH TAKES TWO BOOLEAN AND RETURN A BOOLEAN.

AS A RULE OF THUMB, IF ANY OF THE BOOLEAN HAS A "TRUE", THE RESULT WILL AUTOMATICALLY BE "TRUE" AS WELL, THE RESULT WILL ONLY BE "FALSE" WHEN BOTH BOOLEANS ARE "FALSE".

CONCLUSION

1. There are three basic data types, Number, String and Boolean!

2. These three data types are most commonly used in programming!
3. There are lots of different actions that you can apply to each data type! Explore it on your own!

VARIABLES (VAR)

A LOCAL VARIABLE IS A SYMBOL OR NAME THAT REPRESENTS A VALUE.

INTERACTIVE LESSON: <http://tdev.ly/yrv>

LET'S BEGIN!

Example :

```
| var x := 0
```



First, we declare a variable 'x' and assign a value of 0 to it.

```
| x → post to wall
```



Second, we output the value of 'x' so that the user will be able to see it.

In this case, the output will be 0.

```
| x := 2
```

```
| x → post to wall
```



We can update the value of 'x' by using the `:=` operator and assign a new value to it.

In this case, we assign the value of 2 to 'x' and show the output.

```
| x := x + x
```

```
| x → post to wall
```



We can use variables to add to another variable or itself!

In this case, we add 'x' to itself and update the new value to 'x'.

The output of 'x' will be $2+2 = 4$

```
| var z := x
```



We can create a new variable and assign the value of another variable to the new variable.

In this case, we create a new variable 'z' and assign the value of 'x' to it!

```
| var y := "Hello world "
```

```
| y → post to wall
```



Variables can also be used for Strings or Actions where the action's data can be saved!

In the above example, we create a variable 'y' and assigned a string of "Hello World" to it.

```
| y := x
```

⌚ cannot assign from Number to String



However, do take note that variables of different type cannot be assigned to each other!

In the above example, 'y' has a type of String and 'x' has a type of Number.

CONCLUSION

1. A variable is a symbol or name that represents a value.
2. Values can be of different data type such as Number, String, Boolean (True / False) and more!
3. Variables makes programming more efficient because data can be accessed and updated easily!
4. Variables of different type cannot be assigned to each other!

INVALID VALUES

Here's the link to the tutorial: <http://tdev.ly/qapjc>

CREATING A VARIABLE

To understand what an Invalid Value does, we need to first create a variable and make it ask a number. To do this you have to create a variable then by going to wall, then ask number, you make the program to ask you enter a number.

```
function #0 main ()
```

Let's create a variable and assign the Invalid value

(code of the step)

```
var x := wall → ask number("")
```

Now we need to check if this is an invalid value, so we need to use the function Is Invalid. This returns true or false. It will return true if the variable is invalid and false otherwise. We need to create another variable to store the Boolean value (True/False). To do this create a variable then in the keypad click the variable name of the previous variable you created. For example if you have name the x, then the x should appear on the keypad. Now click x then click Is Invalid.

(code of the step)

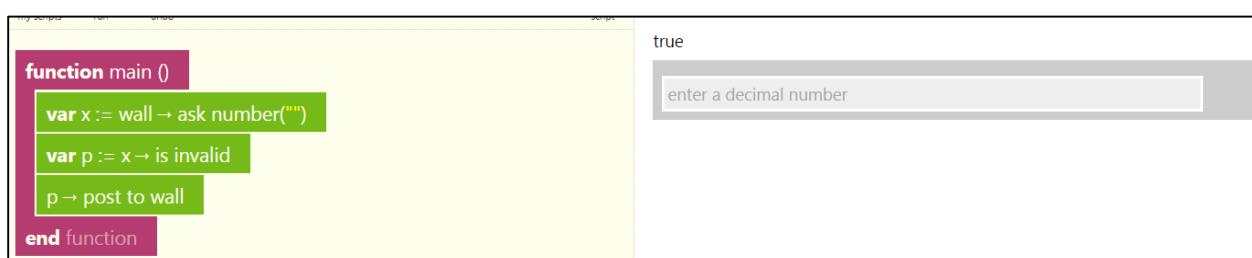
```
var p := x → is invalid
```

Now we need to print this. To do this click the name of the variable in the keypad, then click Post to Wall or you could type it in the search box in the left corner of the screen.

(code of the step)

```
p → post to wall
```

Now run the program by clicking the play button. In the textbook don't enter anything and press OK. If the program displays true then it's working.



Now run the program again, this time enter a value the result should be false.

```

function main()
    var x := wall → ask number("")
    var p := x → is invalid
    p → post to wall
end function

```

The screenshot shows a TouchDevelop script editor window. On the left, a script block contains the following code:

```

function main()
    var x := wall → ask number("")
    var p := x → is invalid
    p → post to wall
end function

```

On the right, the output pane shows the result of running the script. It displays the word "false" above a text input field containing the number "5".

USER INPUT (UI)

USER INPUT ALLOWS US TO GET INFORMATION WE NEED FROM USERS!

INTERACTIVE TUTORIAL : <http://tdev.ly/ubfqa>

We will learn how to get an input from the user using the "Wall" service in TouchDevelop!

User input is an essential part of programming because it makes a program interactive by asking input from the user!

We will take a look on how we are able to request for an input from the user in TouchDevelop!

USING 'ASK STRING'

var name := wall → ask string("What is your name?")

Equivalent output



In the above example, we created a variable called 'name' and assign the 'Wall -> ask string(string input)' service.

The 'name' variable saves the user input so we are able to access it later on in the program!

```
| var greetuser := ("Hello " || name)
```

```
| greetuser → post to wall
```



In the example above, we create a new variable 'greetuser', which concatenate (add) two strings together using the "||" operator.

The result of **greetuser** will be "*Hello 'user's input'*".

USING 'ASK NUMBER'

```
| var age := wall → ask number("What is your age?")
```

```
| var age_output := ("Your age is " || age)
```



Equivalent output

When is your birthday?

enter a decimal number

OK

In this example, we ask the user for his/her age and save it in a variable called 'age'.

We then use the data stored in 'age' to create a new string variable 'age_output'!

USING 'ASK BOOLEAN'

```
var answer := wall → ask boolean("Is programming fun?", "Yes or  
No")
```

Equivalent output

Is programming fun?

Yes or No

NO

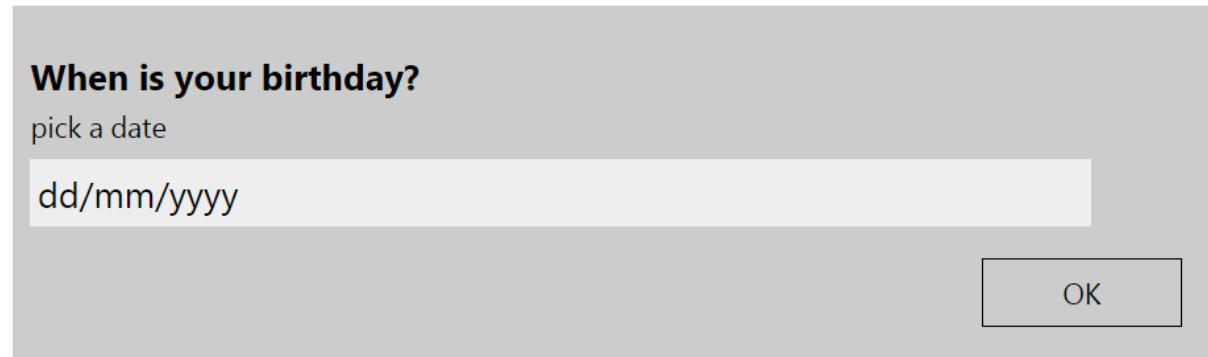
YES

In this example, we ask the user if he/she fun programming fun and the answer is either a 'Yes' or 'No'. However do take note that the value 'answer' stores will be either a 'True' or 'False' and not 'Yes' or 'No'

USING 'PICK DATE'

```
var date := wall → pick date("When is your birthday?", "pick a date")
```

Equivalent output



In this example above, we used the 'pick date' services to get our user to input his/her birthday and we save it in a variable 'date'.

CONCLUSION

1. User Input is important because it makes your program interactive!
2. You can request information that you need from the user!
3. We can choose a variety of actions from 'Wall' to request an information from the user such as:
 - ask string
 - ask number
 - ask boolean (True or false)
 - pick date (Allow user to pick a date in the format of 'mm/dd/yy')
 - pick time
 - pick string (Different from 'ask string')

There are more actions that 'Wall' can provide and you should explore it on your own!

IF AND ELSE IF

This tutorial will teach you about using if and else if statements.

Here is the link to the tutorial for 'if' on TouchDevelop: <http://tdev.ly/ympbwyt>

Here is the link to the tutorial for 'else if' on TouchDevelop: <http://tdev.ly/sbav>

IF STATEMENT

The way that 'IF statements' works, is that it will take a value and see if it fulfills a condition that is chosen and gives the result if it does.

IMPORTANT: The order of the conditions that we use in IF statements are very important. The program will test the conditions in the order that is written, so we must test the more restrictive conditions first.

To use ‘if statements’, firstly, we should enter a value that we want the statement to evaluate. For example, we can assign a variable to equal 15.

```
| var x := 5
```

Next, we will enter the IF statement; lets make the IF statement tell us whether or not the number is greater than 10. We can do it like this:

```
| if x > 10 then  
|   "x is greater than ten!" → post to wall
```

The line of code that is indented and the line below the IF statement tells the program what to do if the condition fulfilled. In this case, when the program is run, the text, “X is greater than 10!” will be posted to the wall.

That’s all well and good, but what if our number was less than or equal to ten? We can then use the second part of the IF statement syntax, called ‘else’. This covers all the cases where the IF statement is not fulfilled.

In TouchDevelop, the ELSE statement is automatically included, and the default code is to ‘do nothing’. For our example, you can add to the code, so that it posts to the wall saying “x is not greater than 10!” like this:

```
| else  
|   "x is not greater than ten!" → post to wall
```

And there we have it, if statements in a nutshell!

ELSE IF

There is also an additional function that can be used alongside IF statements if you want to have more than one condition to test.

The way that ELSE IF statements are used are similar to IF statements that you have learnt already. The only thing that is different is that ELSE IF statements come after the first IF, to test more conditions.

Let’s go through an example to make this clearer. We will write a program that asks the user to input a mark for a test, and the program will post to the wall, the grade that the user inputted.

The grades are as follows: A - 70 and above, B - 55 and above, C - 40 and above, Fail - below 40.

First we need to create the variable to store the mark that the user inputs.

```
| var x := wall → ask number("What is the mark?")
```

We then create the first IF statement as follows:

```
| if x ≥ 70 then  
|   "Wow! That's an A!" → post to wall
```

Because we need other conditions for the rest of the grades, this is where ELSE IF comes in. It is used in the same format as IF, but with the word ELSE included. For the grade B condition, this is how we would implement it:

```
| else if x ≥ 55 then  
|   "That's a B!" → post to wall
```

We can use the ELSE IF statements for the rest of the program as follows:

```
| else if x ≥ 40 then  
|   "That's a C!" → post to wall
```

And for the last statement, you do not need an ELSE IF, you can simply use the ELSE statement, as it will cover the rest of the conditions.

```
| else  
|   "Sorry, that mark is not a pass..." → post to wall
```

That is the basic concept of using IF and ELSE IF statements!

Do the exercises to practice using what you have learnt!

OPERATORS

This tutorial will teach you about operators. Operators are symbols that are used to represent an actions used in programming.

Here is the link to the tutorial on TouchDevelop: <http://tdev.ly/qwausldq>

ASSIGNMENT OPERATOR

Assignment operators are used to assign different types of variables to something. For example, you can assign a number x to equal 1. The symbol we use in TouchDevelop to assign a variable to a value is :=.

Here is an example of what the code would look like on TouchDevelop:

```
var x := 1
```

We can use the same operator to update the value of a variable. For example, we can re-assign the variable x to equal 9.

```
x := 9
```

Note: When we update the value of a variable we do not need to declare the variable again using the ‘var’ feature.

CONCATENATION OPERATOR

Concatenation is when you join two things together.

In TouchDevelop, we use `||` to symbolize the concatenation operator. This takes two values of any type, converts them into string variables and then joins them together.

We can try this out. Firstly, create two different string variables and then use the operator to join the two strings together.

Here is an example that concatenates the two strings “hello” and “world” and posts it to the wall.

```
var s := "hello"  
var s2 := "world"  
(s || s2) → post to wall
```

ARITHMETIC OPERATORS

Arithmetic operators are used to do the simple mathematics operations that we learn. There are 5 main ones that we use in TouchDevelop. They are shown in the table below.

Arithmetic operators operate on number variables and they also return number variables.

Operator	What it means...
+	Adds the number variables together
-	Subtracts the number variables
*	Multiplies the number variables
/	Divides the number variables

math -> mod (x, y)	Calculates the remainder when the number variable x is divided by the number variable y.
--------------------	--

If we create two number variables x and y in the program, we can try these operators. Assign the two variables to any values, for example 9 and 3.

Then, we can add the two numbers together and post it to the wall like this:

```
| (x + y) → post to wall
```

Note: Remember the brackets when using these operators! Otherwise, the computer will get confused, and there will be an error when you run the code.

As with the ‘+’ operator, the ‘-‘ operator can be used in the same format. However, we can assign the result of using an arithmetic operator to a new variable. Look at this example:

```
| var z := (x - y)
```

Here, we have assigned a new variable, z, to the result of $(x - y)$. We can treat this new variable as any other it can be used later on in the code.

The rest of the arithmetic operators are also used in the same way and format as the ‘+’ and ‘-‘ operators. You can practice using these arithmetic operators if you create a new blank script on TouchDevelop.

Additionally, there are a lot of other mathematical operators that can be used. You can find these when you click on the ‘math’ button on the TouchDevelop software.

RELATIONAL OPERATORS

A relational operator is used to compare two different number variables and then returns a boolean value (true or false).

There are 6 different relational operators; they are shown in the table below.

Operator	What it means...
=	Equality - returns true if the two numbers are equal
≠	Inequality - returns true if the two numbers are not equal
≤	Returns true if the first value is less than or equal the second value
<	Returns true if the first value is less than the second value
≥	Returns true if the first value is more than or equal the second value
>	Returns true if the first value is more than the second value

The way that these operators are used is very similar to the arithmetic operators. Here is an example of using the ‘≠’ operator.

($x \neq y$) → post to wall

In the example above, we are comparing the two number variables x and y . In this case, if they are equal, then the program will return ‘false’, but if they are not equal, the program will return ‘true’.

All the relational operators are used in the same way as the ‘≠’ operator, so you can also practice using them in by creating your own blank script.

BOOLEAN OPERATORS

Boolean operators operate on booleans and also return a boolean (true or false). There are 3 boolean operators used with TouchDevelop; they are shown in the table below.

Operator	What it means...
<i>not</i>	This acts as an inverter. For example ‘not clown’. If something is a clown, it will return false. But if something is not a clown, it will return true.
<i>and</i>	This will return true if both conditions are true.
<i>or</i>	This will true if either one or both conditions are true.

These operators are commonly used in if statements, so you will learn more about how to use these in the ‘if’ tutorial.

COMMENTING

Comments are lines of code that contains explanation or notes left by the coder. If a line of code is ‘commented out’ it will not be executed when the program is run.

Comments can be useful when someone who did not write the code is trying to understand what the program does.

To comment out some code in TouchDevelop, you can follow these steps:

1. Click on the line of code that you want to comment out.
2. A bubble that says ‘select’ will appear on the right hand side of the line of code. Click this.

end action

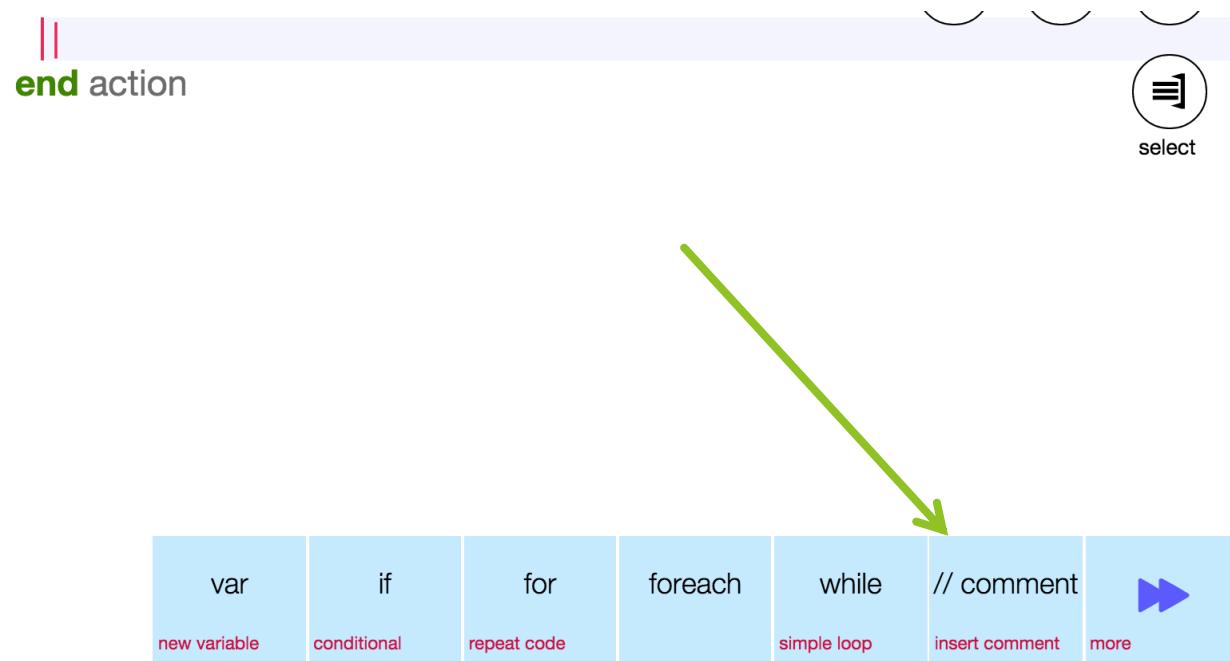


3. A new side tab will appear and at the bottom, you will find a button that says 'comment out'. If you click this, the line of code that you have selected will be commented out with a statement that says 'if false then'.



4. To uncomment a statement, there will be a button that says 'uncomment out'.

You can create a text comment by clicking the '/ /comment' button and then going through the process 1-4 again.



IF AND ELSE IF

This tutorial will teach you about using if and else if statements.

Here is the link to the tutorial for ‘if’ on TouchDevelop: <http://tdev.ly/ympbwyt>

Here is the link to the tutorial for ‘else if’ on TouchDevelop: <http://tdev.ly/sbav>

IF STATEMENT

The way that ‘IF statements’ works, is that it will take a value and see if it fulfills a condition that is chosen and gives the result if it does.

IMPORTANT: The order of the conditions that we use in IF statements are very important. The program will test the conditions in the order that is written, so we must test the more restrictive conditions first.

To use ‘if statements’, firstly, we should enter a value that we want the statement to evaluate. For example, we can assign a variable to equal 15.

```
| var x := 5
```

Next, we will enter the IF statement; lets make the IF statement tell us whether or not the number is greater than 10. We can do it like this:

```
| if x > 10 then  
|   "x is greater than ten!" → post to wall
```

The line of code that is indented and the line below the IF statement tells the program what to do if the condition fulfilled. In this case, when the program is run, the text, “X is greater than 10!” will be posted to the wall.

That’s all well and good, but what if our number was less than or equal to ten? We can then use the second part of the IF statement syntax, called ‘else’. This covers all the cases where the IF statement is not fulfilled.

In TouchDevelop, the ELSE statement is automatically included, and the default code is to ‘do nothing’. For our example, you can add to the code, so that it posts to the wall saying “x is not greater than 10!” like this:

```
| else  
|   "x is not greater than ten!" → post to wall
```

And there we have it, if statements in a nutshell!

ELSE IF

There is also an additional function that can be used alongside IF statements if you want to have more than one condition to test.

The way that ELSE IF statements are used are similar to IF statements that you have learnt already. The only thing that is different is that ELSE IF statements come after the first IF, to test more conditions.

Let's go through an example to make this clearer. We will write a program that asks the user to input a mark for a test, and the program will post to the wall, the grade that the user inputted.

The grades are as follows: A - 70 and above, B - 55 and above, C - 40 and above, Fail - below 40.

First we need to create the variable to store the mark that the user inputs.

```
| var x := wall → ask number("What is the mark?")
```

We then create the first IF statement as follows:

```
| if x ≥ 70 then  
|   "Wow! That's an A!" → post to wall
```

Because we need other conditions for the rest of the grades, this is where ELSE IF comes in. It is used in the same format as IF, but with the word ELSE included. For the grade B condition, this is how we would implement it:

```
| else if x ≥ 55 then  
|   "That's a B!" → post to wall
```

We can use the ELSE IF statements for the rest of the program as follows:

```
| else if x ≥ 40 then  
|   "That's a C!" → post to wall
```

And for the last statement, you do not need an ELSE IF, you can simply use the ELSE statement, as it will cover the rest of the conditions.

```
| else  
|   "Sorry, that mark is not a pass..." → post to wall
```

That is the basic concept of using IF and ELSE IF statements!

Do the exercises to practice using what you have learnt!

LOOPS (FOR AND WHILE)

MOVING OBJECT

To get you started, we're going to begin with how to make an object moves. Let say in this case we want to make a clown moves.

(code of the step) This is to make the clown facing left, the number inside the bracket is the angle of Clown's movement to the left

- ④ clown → left turn(45)
- ④ clown → forward(100)

Now try this one on your own! The clown will now moves in 45 degree to the left with the length of 100 moves. We are now wondering, if we want to make multiple moves, we need to write a lot of code to place the clown to move from one place to another, individually. But lucky for us, we have a shortcut called Loops.

There are different kind of loops, such as for and while. Let's start off with for loop.

FOR LOOP

Here is the link to the tutorial on TouchDevelop. <http://tdev.ly/euxle>

For statement can be used to execute a line of code into a desired number of times. For example, rather than commanding the clown to moves 3 times to make an octagon, we can just use for loop and perform it 8 times:

This loop is to perform the action all over again

for $0 \leq i < 8$ **do**

The steps are split using a depth-first-search strategy.

- ④ clown → left turn(45)
- ④ clown → forward(100)

Run your program: This is how the clown moves

end for

In this example, 'i' is acting as a random variable just to store how many time we want to perform the code. Now try it by yourself in your tablets!

WHILE LOOP

Here is the link to the tutorial on TouchDevelop. <http://tdev.ly/acrfq>

We can implement the exact same thing as for loop by using a while loop. The only difference is while loop is using a stored variable on top of the actual loop. We'll start introducing the variable, then perform the while loop

```
|| var index := 7
  ||| while index ≥ 0 do
    |||| ⚙ turtle2 → left turn(45)
    |||| ⚙ turtle2 → forward(100)
    |||| index := index - 1
    ||| end while
  || end action
```

Whoops, but wait, we want an 8-sided trail, why is it only 7? Well, take a look at the symbol there. We start counting from 0 instead of 1. Now try it by yourself and try out countless shapes that you can think of!

PART III - MEET THE ENGDUINO

INTRODUCING THE ENGDUINO

What is the Engduino? Well, it's a nifty little computer that you can program to do the things that you want it to do, with an array of sensors, LED's and other little gizmos to play around and do cool things with.

Without further ado, let's introduce the Engduino (the Engduino 3, to be precise) and some of its hardware.

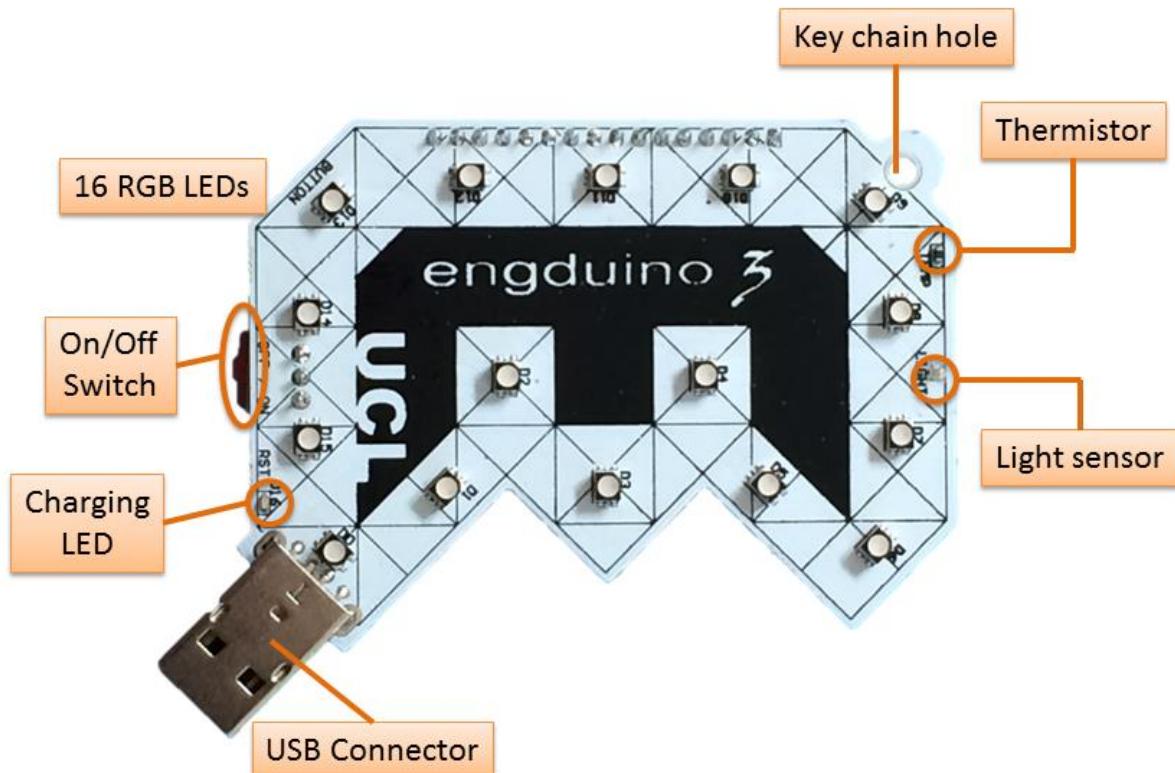


Figure 2: The front side of the Engduino V3. Neat.

On the front side of the Engduino you have:

- A set of 16 LED's that can light up in pretty much any colour you want them to.
- A key chain hole, if you're into that sort of thing.
- A thermistor, for taking temperature readings.
- A light sensor, for taking readings of how much light is falling on it.
- An LED that shows if the Engduino battery is charging.
- A USB plug for sticking the Engduino into your PC or Mac.
- An On/Off switch, which I really shouldn't have to explain to you.

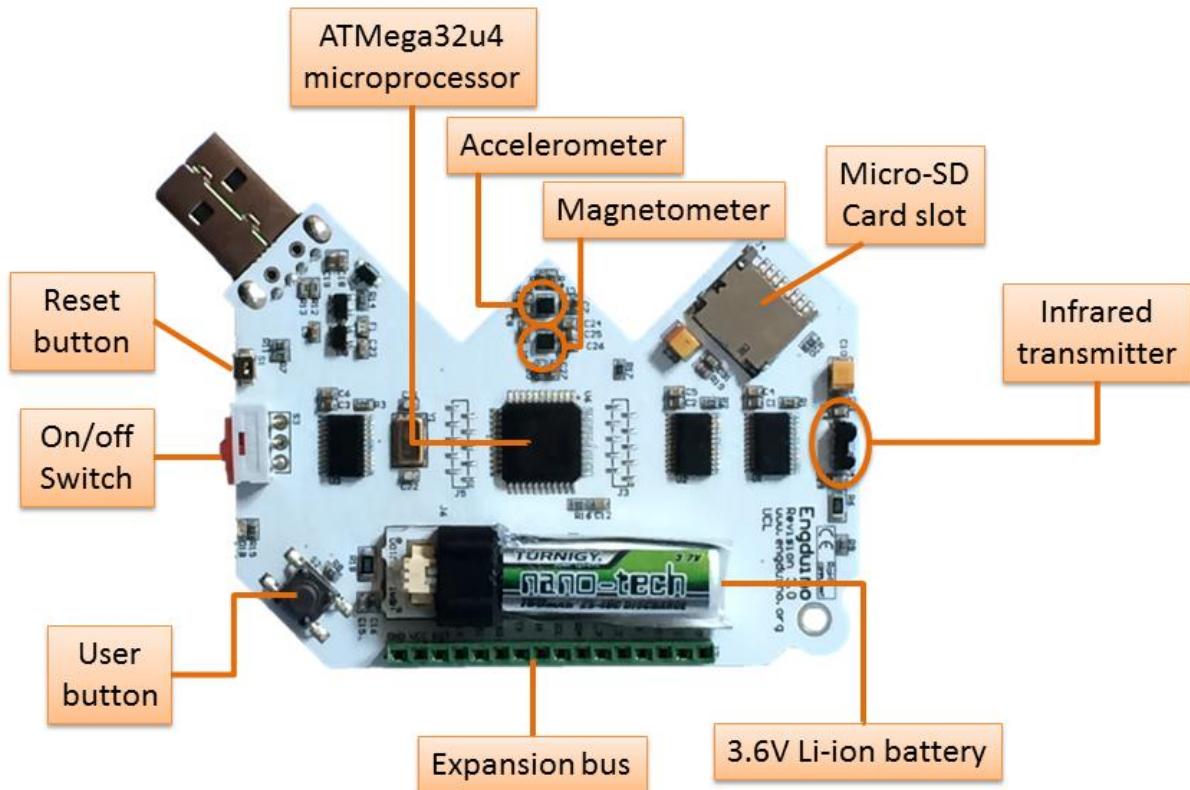


Figure 3: The rear side of the Engduino V3. It's a bit messy back here.

Flip the Engduino over and you'll find:

- The processor and the other weird technical stuff.
- A Micro-SD card slot for Micro-SD cards (and nothing else!)
- A 3.6V Lithium-Ion battery to power the thing.
- A user button. What does it do? That's for you to decide.
- A reset button. This resets the Engduino to its factory settings, don't press this unless you really need to, seriously.
- An infrared transmitter to transmit infrared stuff.
- An accelerometer to measure the acceleration of the Engduino.
- A magnetometer to measure magnetic field strength, perfect for calibrating the magnets on your giant city-destroying railgun.
- An expansion bus, for interfacing with other hardware.

WHAT IS THE ENGDUINO USED FOR?

The Engduino is a great toy for beginner programmers to play around with. The variety of sensors and other features can be used in various different ways, and it's good fun to come up with new things for it to do. It's excellent programming practise, a lot more interesting than mucking about with dusty old text output on a computer, and a good way of being introduced to the more practical ways of programming.

Here are some of the great, life-changing things that you could do with the Engduino:

- Have the LED's light in sequence to form a timer.
- Have the LED's change colour depending on the temperature read by the thermistor.
- Turn the Engduino into a spirit level, with the LED's lighting a pattern representative of the Engduino's current orientation.
- Make your weekends less boring by turning the Engduino into a budget disco machine; where all the LED's will flash randomly in randomised colours.

There are of course many other things that you could do, the only limit is your creativity (and of course, the Engduino itself). You could modify some of the ideas above to incorporate other pieces of the Engduino's hardware, such as having the LED timer stop when the button is pressed, for example.

FOR AND WHILE LOOPS

Let's try and implement some of the programming knowledge we learned earlier with the Engduino, specifically For and While loops.

FOR LOOP

Let's begin with a For loop - Since each LED on the Engduino has its own unique ID, we can refer to them via each iteration of a For loop. In this example, we'll make each LED light in sequence, with all the lights turning off at the end:

action main ()

```
This action emulates the code loop in the bug and will not be compiled.  
do nothing  
▷ setup  
for 0 ≤ i < 17 do  
    ⏚ engduino → set LED(i, colors → blue)  
    ⏚ time → sleep(1)  
end for  
    ⏚ time → sleep(1)  
    ⏚ engduino → set all LEDs(colors → black)  
end action
```

Let's break it down:

- We set up a For loop for when i, our For loop's local counter variable, is less than 17, meaning that the loop will execute for each value of i up to **and including** 16.
- In the body of the For loop, the Engduino will illuminate the LED with the ID of the current value of i, then sleep for a second (so that the lights illuminate at a noticeable pace).
- This process repeats itself until i = 17, at which point all 16 LEDs will be illuminated, and the For loop will end.
- After the for loop ends, all of the LEDs will turn off.

Now try modifying this example, here are some exercises to get you started:

1. Make the entire process repeat itself. I.e. when the LEDs all turn off, they will begin lighting again, and so on.
2. Turn the Engduino into a two-part minute timer: with ten of the LED's representing seconds, and the remaining six representing tens of seconds. You'll need to use nested For loops for this one.

WHILE LOOP

This example will be a bit different: instead of having each LED turn off (and stay on) in sequence, we will have a perpetual While loop check if the Engduino's button has been pressed, turning the LED's on (or off) if it is pressed:

action main ()

This action emulates the code loop in the bug and will not be compiled.

do nothing

▷ setup

var lights := 0

do nothing

while true do

⌚ **if** ⚖ engduino → button was pressed **then**

if lights = 0 **then**

lights := 1

⌚ engduino → set all LEDs(colors → blue)

else if lights = 1 **then**

lights := 0

⌚ engduino → set all LEDs(colors → black)

else do nothing **end if**

else do nothing **end if**

end while

⌚ engduino → set all LEDs(colors → black)

end action

Let's break it down:

- We initialise a variable to keep track of whether the LEDs are currently on or off.
- We set up a While loop with true set as the argument. What this will do is making the While loop to repeat indefinitely, if/until a statement in the loop body cancels (breaks) it.
- Each time this loop executes, it will check to see if the Engduino's user button has been pressed.
- If it has, it will change our LED variable to 0 if it is currently 1, or to 1 if it is currently 0, and turn the LEDs on or off accordingly.
- The only way to terminate the While loop in this instance would be to turn off the Engduino.

Now try modifying this example, here are some exercises to get you started:

1. Have pressing the button change the colour of the LEDs change if they are already turned on.
2. If you're up for a challenge; integrate the previous exercise into this one and create a program where the For loop in the previous exercise will be terminated if the Engduino button is pressed. I.e. the LEDs will turn on in sequence until the button is pressed. Good luck!

BUTTON

And now let's learn about how to use the button that's on the back of your engduino, you can basically implement the button to any of your project, so let's get started:

FUNCTIONS

1. Button pressed
2. Button was pressed

BUTTON PRESSED

This function tells the user if the button is currently pressed. It returns a Boolean value. It returns true if the button is pressed, false if it is not pressed. For example the program below has an if statement. It will execute if the button is pressed and the LED's colour will turn to red. If the button is released the if statement condition will become false so the LED's will turn off by setting the colour to black.

```
while (true) {
    if (engduino → button pressed) {
        engduino → set all LEDs(colors → red)
    } else {
        engduino → set all LEDs(colors → black)
    }
}
```

BUTTON WAS PRESSED

This function tells the user if the button was pressed. It returns a Boolean value. For example in the program below if the button is pressed then the LED's will turn to red and won't turn off like the button pressed function.

```
while (true) {  
    if (engduino → button was pressed) {  
        engduino → set all LEDs(colors → red)
```

Exercises

1. Make the Engduino to change the colour when the button is pressed twice
2. Make the Engduino to set a colour if the button is pressed

LEDS

Now we're going to learn about to set up colors of the LEDs that will be displayed on your engduino

SET COLOUR OF ALL LEDS

You can set the colour of all the LED's at the same time, by using 'set all LED'.

Here is an example where the colour is random, using TouchDevelop:



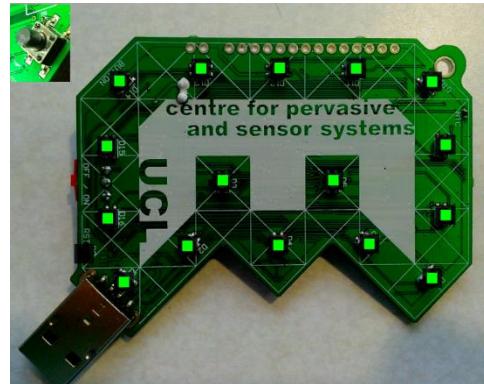
```
|| engduino → set all LEDs(colors → random)
```

To set the LEDs to a specific colour, instead of 'random', change this to the other colours available that are on TouchDevelop, which are **red, green, white, blue, orange, dark grey, grey, light grey and yellow**.

E.g. Setting all the LEDs to green.



Engduino → set all LEDs(colors→ green)



Set the colour to **black** to switch the LEDs OFF.

SETTING A SINGLE LED

You can set the colour of a single LED, by using ‘set LED(*LED number, colour*)’.

The number of the LED is labelled on the Engduino, from 1-16. Note: LED 0 is on the side of the Engduino.

Here is an example where LED 1 is set to red:



Engduino → set LED(1, colors→ red)

You can also use the ‘colors → from rgb(r, g, b)’ function to use other colours. Here is the link, where you can find the RGB values for different colours:

http://www.rapidtables.com/web/color/RGB_Color.htm

USING LOOPS

If you wanted to set the odd numbered LEDs to red and even numbered LEDs to green, then instead of listing out each one individually, you can use loops!

Here is how you would do this:

```

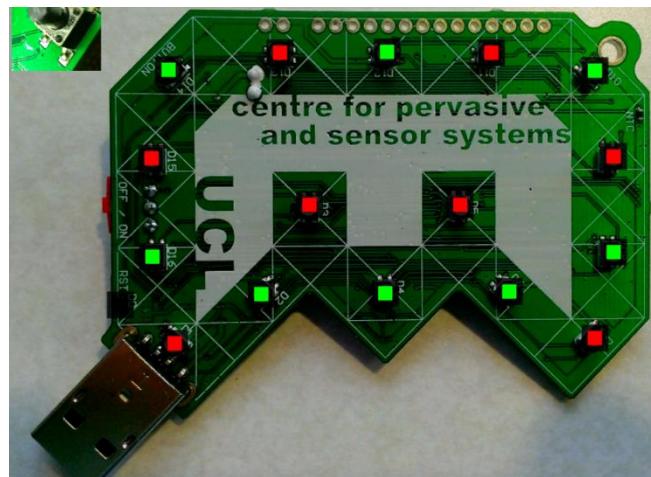
for 0 ≤ i < 17 do
    Engduino → set LED(0, colors → black)
    if math → mod(i, 2) = 0 then
        Engduino → set LED(i, colors → green)
    else
        Engduino → set LED(i, colors → red)
    end if
end for

```

The ‘for’ loop always starts at 0, and it goes up to 17 so that it includes all 16 LEDs. We first set LED 0 to black, as we do not want this one to be on (it is the small LED on the side).

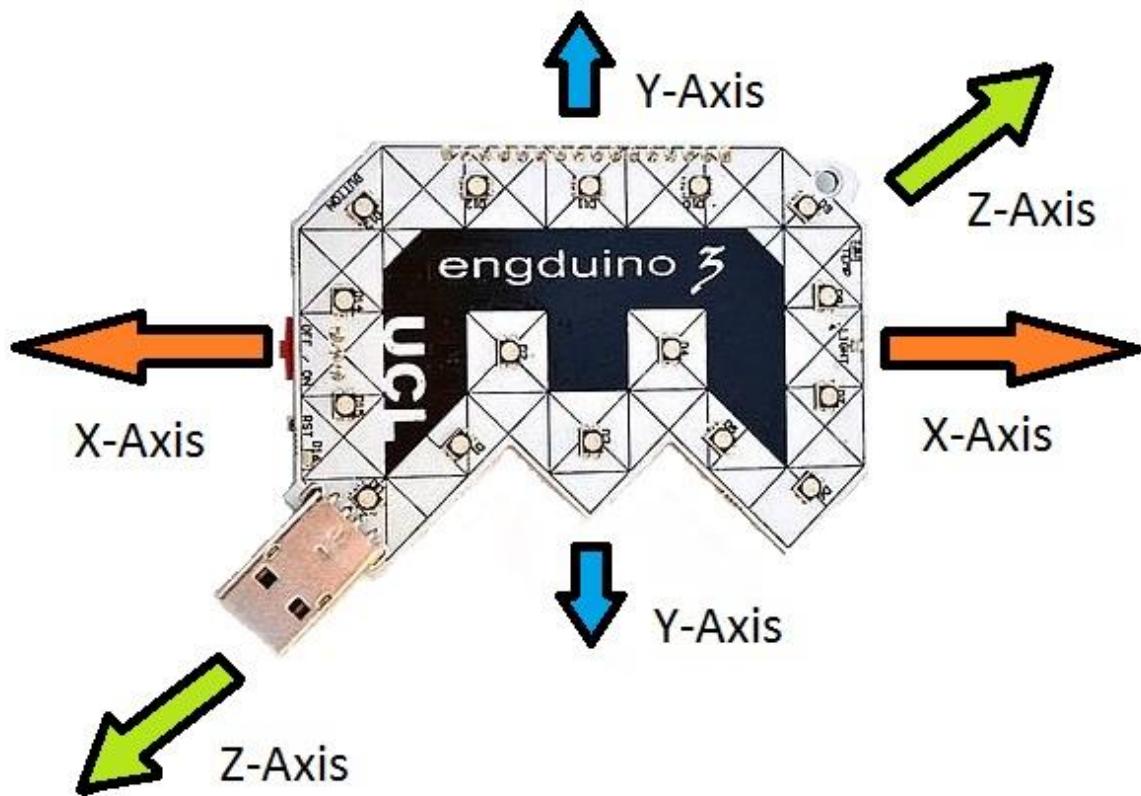
We use the ‘math → mod’ function and an ‘if statement’ to separate the odd even and odd numbers (the remainder of an even number divided by 2 would equal to 0, whereas odd number divided by 2 would give a remainder).

Here is the result:



ACCELEROMETER

The accelerometer on the Engduino measures the acceleration on the Engduino. It is able to measure acceleration in three dimension (In 3-Axis known as X,Y,Z)



In this Engduino lesson, we will be covering the following point,

- Storing the accelerometer readings into a variable.
- Read and understand the accelerometer readings in Engduino.
- Using the accelerometer readings to perform different task.

STORE ACCELEROMETER READINGS

In order for us to read the accelerometer readings, we need to store the accelerometer readings into a variable first.

while true do

 ▷ loop

(code of the step) We will store the accelerometer value into a variable

 🕒 **var** axis := 🌐 engduino → acceleration

end while

In the codes above, we stored the engduino accelerometer readings into a variable called 'axis'

*Note that the variable 'axis' will store 3 different values(X,Y,Z) with one single line of code.

READ ACCELEROMETER READINGS

To get the readings of the Engduino, we simply output the variable containing the readings of the accelerometer.

var axis := 🌐 engduino → acceleration
axis → post to wall

Output

(0.0168469,-0.124751,0.991898)
(0.0156674,-0.121104,0.992375)
(0.00000,1.00000,0.00000)
(0.00000,1.00000,0.00000)
(X value, Y value, Z value)

When you run the codes, you will get the above output. The readings shows the Engduino accelerometer readings in term of (X , Y , Z). The highest and lowest value that X,Y,Z can get is ±1.

*Note that the Y-axis reading is 1.0 by default? That's because of gravity acting downwards relative to the Engduino.

We can also read the value of X, Y, Z individually using the codes below

```
var axis := engduino → acceleration  
axis → x → post to wall  
axis → y → post to wall  
axis → z → post to wall
```

USING ACCELEROMETER READING TO PERFORM AN ACTION

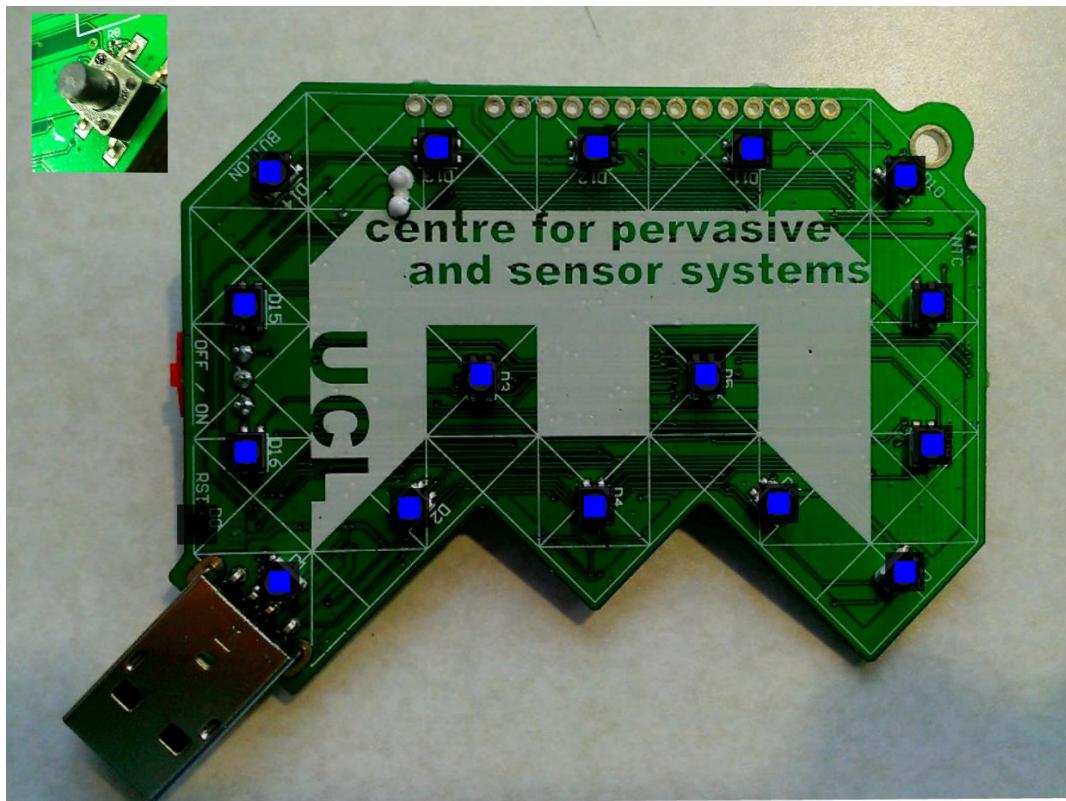
Now that we know how to get the accelerometer readings, we will use the readings along with 'if/else' statement to perform different actions.

while true do

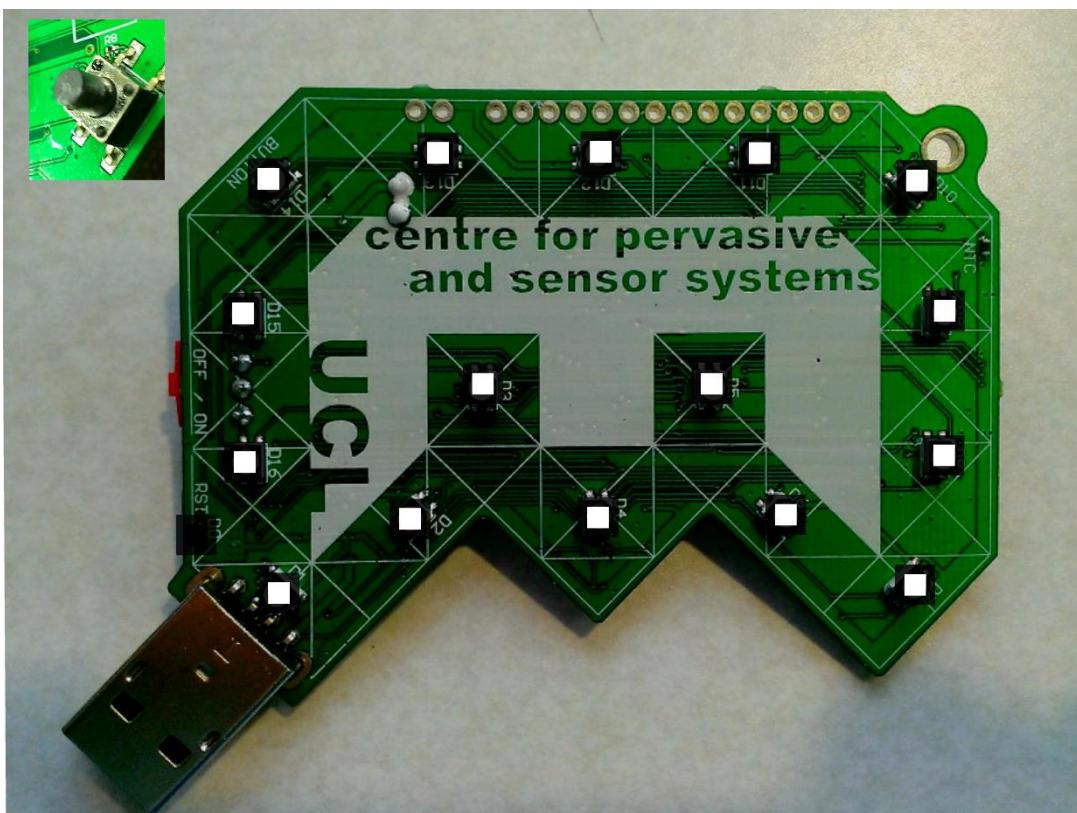
```
▷ loop  
(code of the step) We will store the accelerometer value into a variable  
var axis := engduino → acceleration  
if axis → x > 0.5 then  
    engduino → set all LEDs(colors → blue)  
else if axis → x < - 0.5 then  
    engduino → set all LEDs(colors → red)  
else  
    engduino → set all LEDs(colors → white)  
end if  
end while
```

In the code above, we use the accelerometer X-axis reading to change the LEDs colour on the Engduino. The LEDs will become blue when the accelerometer X-axis reading is above 0.5.

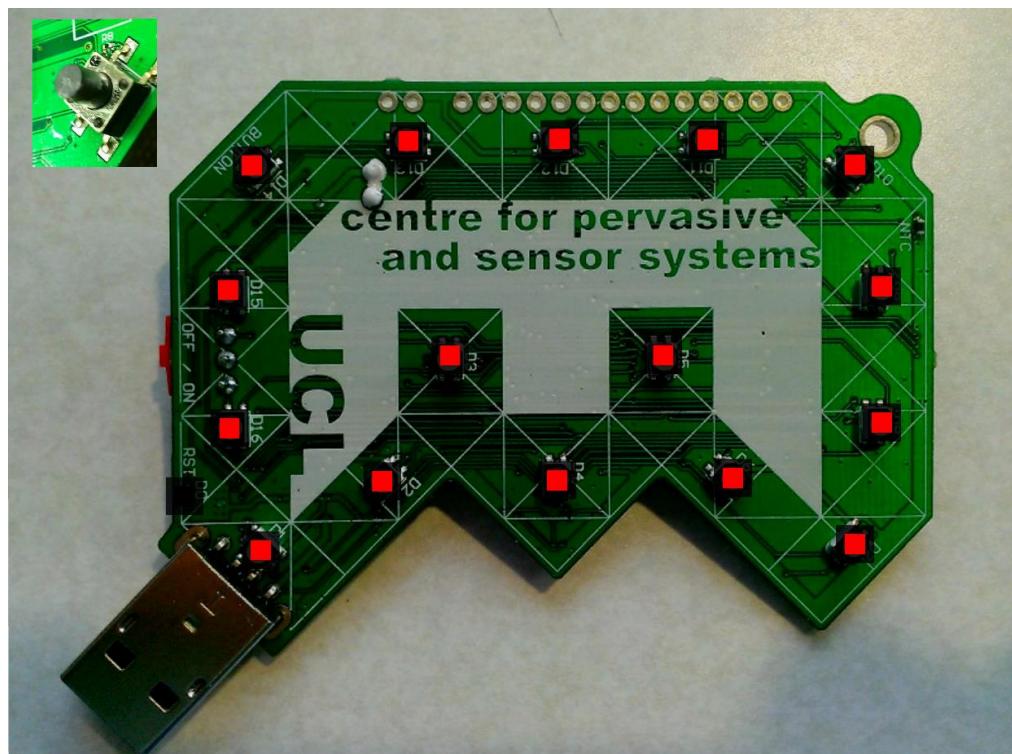
If the accelerometer X-axis reading is between -0.5 and 0.5 ($-0.5 \leq X \leq 0.5$), the LEDs will become white, and if the X-axis reading is below -0.5, the LEDs will become red.



When X-axis reading is above 0.5 (Move right)



When X-axis reading is between -0.5 and 0.5



When X-axis reading is below -0.5 (Move left)

CONCLUSION

1. Accelerometer on the Engduino measures three dimension of acceleration.
2. Readings stored in a variable can be shown as (X, Y, Z) or as individual axis reading.
3. Using individual axis-reading, we can perform a variety of actions.

SUMMARY GAME | WORLD CUP FLAGS

Who doesn't love the world cup? In this mini quiz, we're going to combine all of the Engduino exercises before (Accelerometer, LEDs, loops, etc) to make an awesome Engduino world cup flags display. Let's four of your favorite countries and display their country flag into our Engduino pad LEDs. Let's say these are your favorite teams from world cup: Spain, France, Italy and England.



To display four of the flags, let's set up Spain flag when we're tilting to the left, Italy to the top, France to the right and England to the bottom.

SETTING UP THE LED

The First step is to set up the LEDs for all four countries. Let's take Spain flag as the example. In order to make a Spanish flag, we need two different LEDs colors which is yellow at the middle and two arrays of reds at the top and bottom. In this case, we're going to take Engduino green pad as the example (since every Engduino pads has different numbering system).

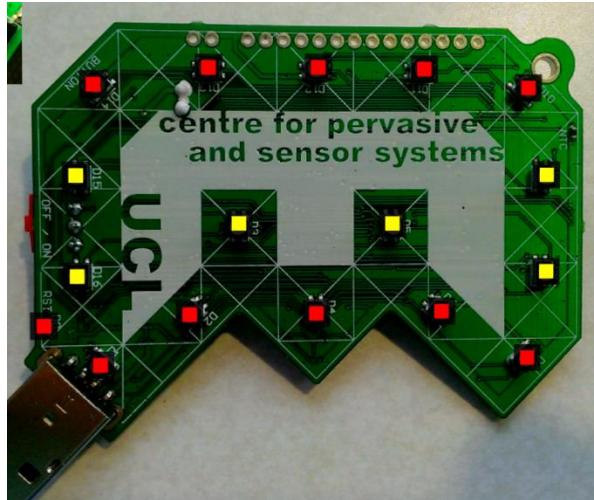
Let's set up LEDs number 1,2,4,6,7 and 0 and LEDs number 9 to 15 using if condition into red lighting and the rest to yellow. Maybe some of you are wondering, why can't we use for loop in this case. In other language, indeed we can use for loop to set up LEDs from 9 to 15 but since TouchDevelop can only support loop that starts from 0, we have to use if condition as the substitute. Let's take a look at the real code:

```

for 0 ≤ i < 17 do
    if i = 1 or i = 2 or i = 4 or i = 6 or i = 7 or i = 0 then
        ↗ engduino → set LED(i, colors → red)
    else if i > 9 and i < 15 then
        ↗ engduino → set LED(i, colors → red)
    else
        ↗ engduino → set LED(i, colors → yellow)
    end if
end for

```

As you can see, we can use if condition to make the code more simple than writing it one by one. As we expected, this is the display of the Engduino on the emulator:



COMBINING THE LEDs AND ACCELEROMETER

Now, the second step is when you tilt the Engduino to the right, it shows the Spanish flag, and leave the rest to black. The way to do it is very simple, just wrap all of our code that we just made with if statement.

```

action main ()
  This action emulates the code loop in the bug and will not be compiled.
  while true do
    var axis := Engduino → acceleration
    if axis → x > 0.5 then
      for 0 ≤ i < 17 do
        if i = 1 or i = 2 or i = 4 or i = 6 or i = 7 or i = 0 then
          Engduino → set LED(i, colors → red)
        else if i > 9 and i < 15 then
          Engduino → set LED(i, colors → red)
        else
          Engduino → set LED(i, colors → yellow)
        end if
      end for
    else
      Engduino → set all LEDs(colors → black)
    end if
  end while
  ▷ setup
end action

```

When you're testing it to the Engduino, you can see that the flag only shows up when we tilt the pad to the right and when it's down to any other direction, all of the LEDs are off.

MULTIPLE FLAGS!

So, how about adding another flags to the pad?

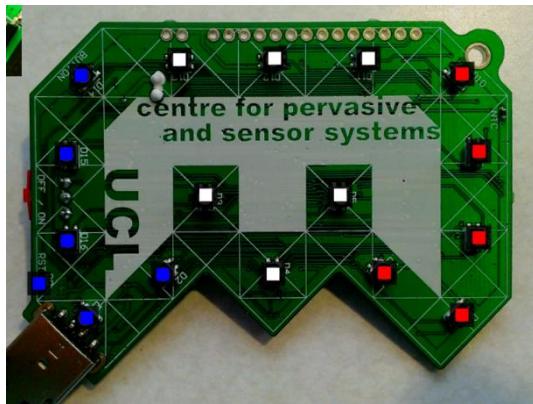
We're going to visit our old pal else if in this one. Let's say we want to add the French flag when we want to tilt to the left. At the end of our previous code, instead of just closing it with else, we can add else if to input our French flag code. Here's how it goes:

```

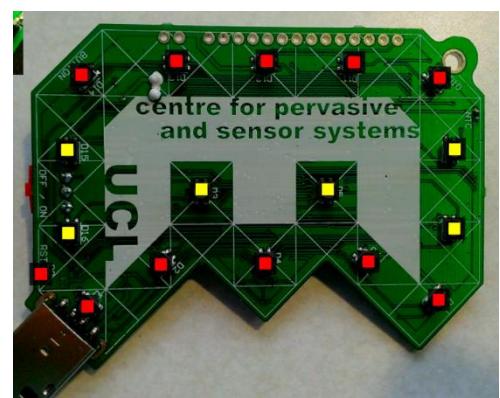
action main()
This action emulates the code loop in the bug and will not be compiled.
while true do
    var axis := engduino → acceleration
    if axis → x > 0.5 then
        for 0 ≤ i < 17 do
            if i = 1 or i = 2 or i = 4 or i = 6 or i = 7 or i = 0 then
                engduino → set LED(i, colors → red)
            else if i > 9 and i < 15 then
                engduino → set LED(i, colors → red)
            else
                engduino → set LED(i, colors → yellow)
            end if
        end for
    else if axis → x < -0.5 then
        for 0 ≤ j < 17 do
            if j = 1 or j = 2 or j = 0 then
                engduino → set LED(j, colors → blue)
            else if j > 13 and j < 17 then
                engduino → set LED(j, colors → blue)
            else if j > 5 and j < 11 then
                engduino → set LED(j, colors → red)
            else
                engduino → set LED(j, colors → white)
            end if
        end for
    else do nothing end if
end while
▷ setup
end action

```

As you can see, this code contains two different flag tilts, here's the final result on the emulator:



Tilt to the left



Tilt to the right

QUICK EXERCISES

1. Now you can finish up all the four sides with the rest of the flag by using all the statements mentioned before!
2. Add the button function so that the lights only emit when the button is pressed

FINAL PROJECT

ENGDUINO RACE (STEP COUNTER)

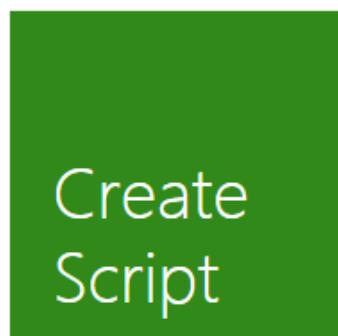
In this tutorial, we will teach you to write the code to make an Engduino race meter! Let's call our little Engduino race meter a Pedometer.

Our Pedometer will count the number of steps the user take and light up the Engduino's LED according to the number of steps he/she takes! So at the end of the race you and your friends can take a healthy run then compare to each other who wins the race by counting the total of LEDs lights that lit!

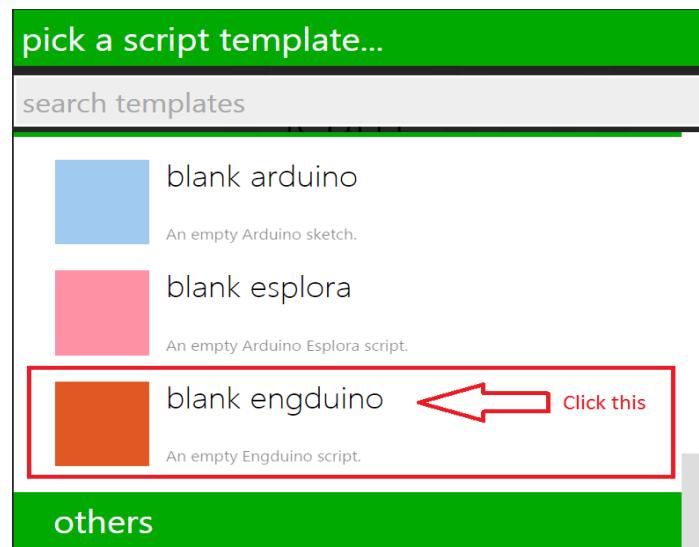
We are able to use the Engduino to create a simple Pedometer by using the accelerometer on the Engduino. The Pedometer counts the number of steps the user takes by calculating the acceleration from the Engduino.

GETTING STARTED!

Before we can create our Pedometer, we have to set up the script first!



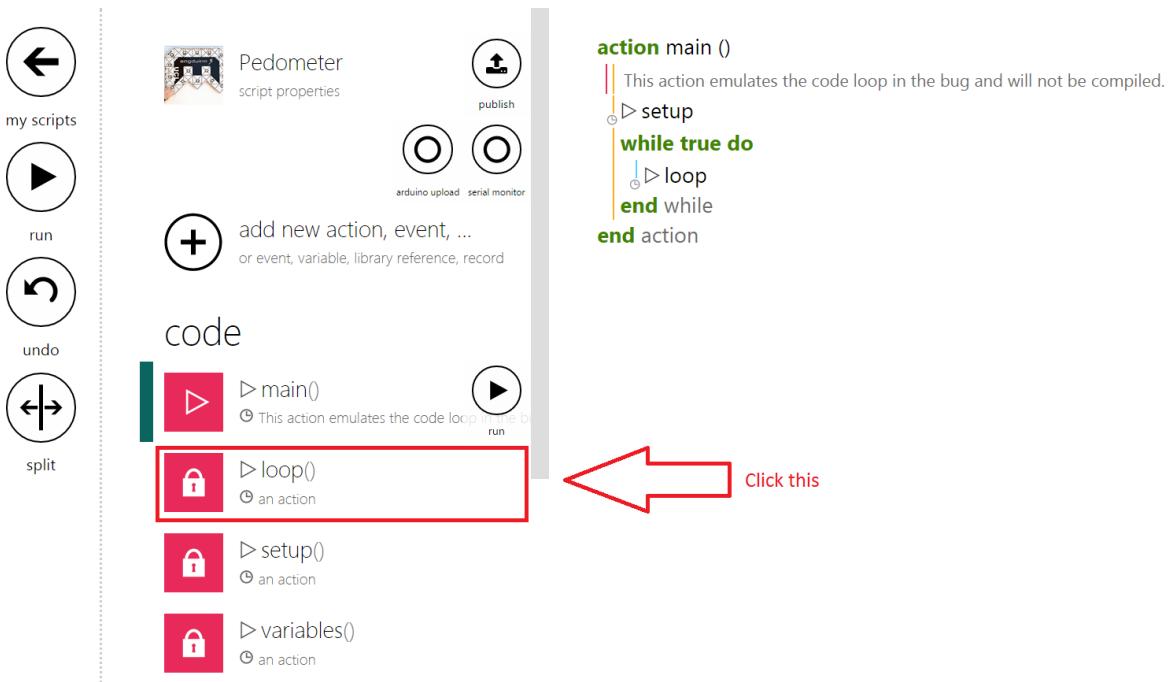
First, click on "Create Script"



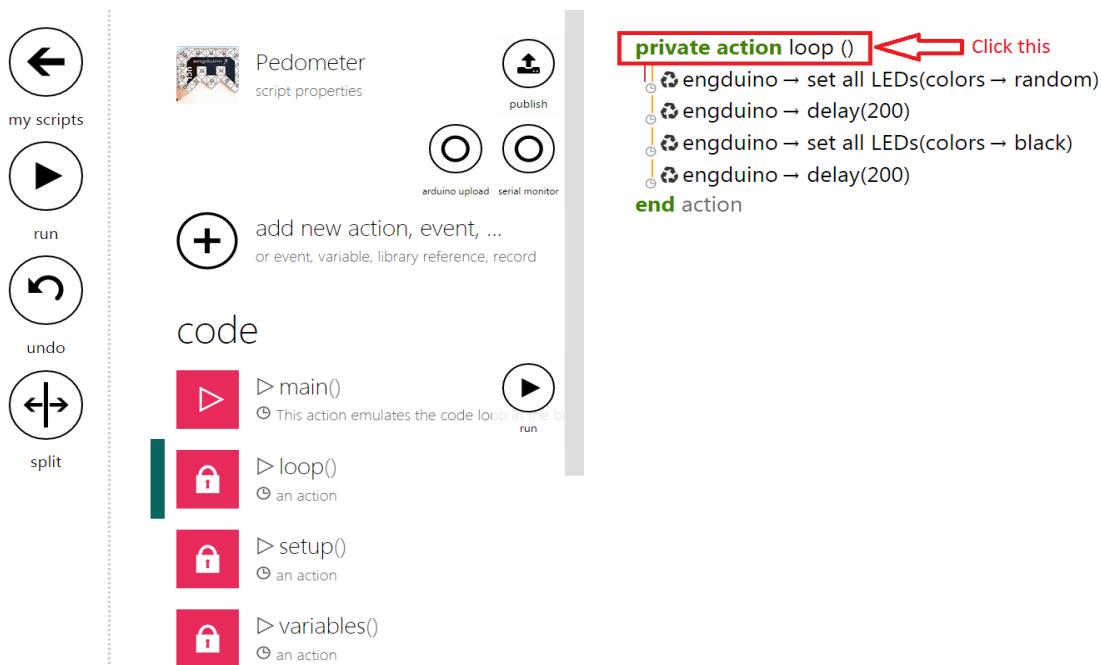
Next, click on "blank engduino"

A screenshot of a new script creation interface. The title "blank engduino" is displayed in large white text on a blue background. Below it is the subtitle "An empty Engduino script.". A text input field contains the text "Pedometer". In the bottom right corner is a white button with the word "create" in black text.

Name your new script as "Pedometer" and click create



This is the main page of your script! But before we start coding, we need to delete some codes from the template!



Click on the "private action loop()" to bring up the menu to delete the loop function.

action

loop

private action
Private actions do not get a run button.

'atomic' action [more info](#)

add input parameter
create a new parameter

add output parameter
create a new parameter

cut copy

delete Click this

Ready for more options? [Change skill level!](#)

Click on "delete" to delete the function.

my scripts

run

undo

split

Pedometer script properties

publish

arduino upload serial monitor

code

main()
This action emulates the code loop in the bug and will not be compiled.
▷ setup
 || ▷ → loop
 || i cannot find property 'loop' on code [TD112]
end while
end action

setup()
an action

variables()
an action

You will be redirected to the Main function. Click on loop statement and delete it from the codes.

LET'S START CODING OUR PEDOMETER!

```
action main ()
```

```
    ▷ setup
```

```
    var counter := 0
```

We declare a counter variable to count the number of steps the user has taken in total

Create a variable and rename it to "counter" and initialise it to 0.

```
action main ()
```

```
    ▷ setup
```

```
    var counter := 0
```

We declare a counter variable to count the number of steps the user has taken in total

```
while true do
```

```
    var p := engduino → acceleration
```

Next we create a variable "p" and assign it with the Engduino's accelerometer!

```
action main ()
```

```
    ▷ setup
```

```
    var counter := 0
```

We declare a counter variable to count the number of steps the user has taken in total

```
while true do
```

```
    var p := engduino → acceleration
```

```
    var x := p → x
```

```
    var y := p → y
```

```
    var z := p → z
```

We assign 3 different variables to store our X,Y and Z values of the Engduino's Accelerometer

Next, we store the accelerometer's X,Y and Z readings to 3 variables.

```

action main ()
|▷ setup
|  var counter := 0
| We declare a counter variable to count the number of steps the user has taken in total
| while true do
|   var p := engduino → acceleration
|   var x := p → x
|   var y := p → y
|   var z := p → z
| We assign 3 different variables to store our X,Y and Z values of the Engduino's Accelerometer
|   var acceleration := x * x + y * y + z * z
|   acceleration := math → sqrt(acceleration)
| In order to calculate the acceleration from the Engduino, we have to sum the square of x,y,z and square root
| the summation

```

We create a new variable "acceleration" which sums the square of X,Y and Z.

We then square root the summation to get the real acceleration.

The acceleration formula is given as :

$$\text{Acceleration} = \sqrt{x^2 + y^2 + z^2}$$

The acceleration formula calculates the user's acceleration by using all 3 axis of the accelerometer to determine if the user has taken a step

*Note that this calculation is not the real way of how an actual pedometer calculates a step!

```

action main ()
  ▷ setup
    var counter := 0
    We declare a counter variable to count the number of steps the user has taken in total
  while true do
    var p := engduino → acceleration
    var x := p → x
    var y := p → y
    var z := p → z
    We assign 3 different variables to store our X,Y and Z values of the Engduino's Accelerometer
    var acceleration := x * x + y * y + z * z
    acceleration := math → sqrt(acceleration)
    In order to calculate the acceleration from the Engduino, we have to sum the square of x,y,z and square root
    the summation
    if acceleration ≥ 1.1 then
      counter := counter + 1
      For us to register a step, we have to make sure the acceleration from the user is at least of acceleration 1.1

```

In order for us to know if a user has taken a step, we have to check if the acceleration of the user is above 1.1g. If the acceleration is above 1.1g, we consider it as a step and we add the step to the counter!

CREATING LED OUTPUT ACCORDING TO NO. OF STEPS TAKEN.

The screenshot shows the Tinkercad interface with the 'code' tab selected. On the left, there's a list of actions: main(), setup(), and variables(). A red box highlights the '+ add new action, event, ... or event, variable, library reference, record' button, with a red arrow pointing to it labeled 'Click this'. On the right, a sidebar titled 'create a new...' shows search results for 'action()'. The first result, 'action()' with the sub-description 'Code that performs a specific task', is also highlighted with a red box and a red arrow pointing to it labeled 'Click this'.

We will add a new action to show the output of the counter on the Engduino's LED

The screenshot shows the 'action' configuration panel. On the left, there are icons for dismiss, run, undo, and split. The main area has a title 'action' and a sub-section for 'StepCounter'. It includes a checked checkbox for 'private action' and an unchecked checkbox for "'atomic' action". Below these are buttons for 'add input parameter' and 'add output parameter'. At the bottom are buttons for 'cut', 'copy', and 'delete'. Three red arrows point from the text 'We will add a new action to show the output of the counter on the Engduino's LED' to the 'private action' checkbox, the 'add input parameter' button, and the 'add output parameter' button.

We will rename our action to "StepCounter" and make it a private action and also add an input parameter.

```
private action StepCounter (
    step : Number)
do
    for 0 ≤ i < 17 do
        if i = 0 then
            i + 1
            ↪ '+' returns a 'Number'; insert 'post to wall' if you want to display it
        else do nothing end if
        engduino → set LED(0, colors → black)
```

We rename our input parameter to "step" (This input will be the *counter* from the main function!) and include a "for" loop with a range of 0 to 16 because we want to light up all 16 LEDs on the Engduino.

We also include a statement to make *i*=1 as the LED output starts from 1 and not 0.

```
private action StepCounter (
    step : Number)
do
    for 0 ≤ i < 17 do
        if i = 0 then
            i + 1
            ↪ '+' returns a 'Number'; insert 'post to wall' if you want to display it
        else do nothing end if
        engduino → set LED(0, colors → black)
        var count := i * 20
        if step > count then
            ↪ engduino → set LED(i, colors → green)
            i + 1
            ↪ '+' returns a 'Number'; insert 'post to wall' if you want to display it
```

Next, we create a new variable called 'count' and initialize it to be the number of steps we want the user to take so that a LED will light up. In this case, we want the user to walk 20 steps before lighting one LED up.

In order for all 16 LEDs to be lighted up on the Engduino, the user has to walk :

16 * No. of steps you want the user to take*

So in the case of our code, the user has to walk 320 steps so that all the LED would light up.

*You can assign any value you want the user to walk before lighting up an LED!

```
private action StepCounter (
    step : Number)
do
    for 0 ≤ i < 17 do
        if i = 0 then
            i + 1
            ↳ '+' returns a 'Number'; insert 'post to wall' if you want to display it
        else do nothing end if
        engduino → set LED(0, colors → black)
        var count := i * 20
        if step > count then
            ↳ engduino → set LED(i, colors → green)
            i + 1
            ↳ '+' returns a 'Number'; insert 'post to wall' if you want to display it
            if i = 16 then
                ↳ engduino → set all LEDs(colors → black)
                ↳ engduino → delay(500)
                ↳ engduino → set all LEDs(colors → green)
                ↳ engduino → delay(500)
            else do nothing end if
        else do nothing end if
    end for
end action
```

beta 80345 © 2015 Microsoft

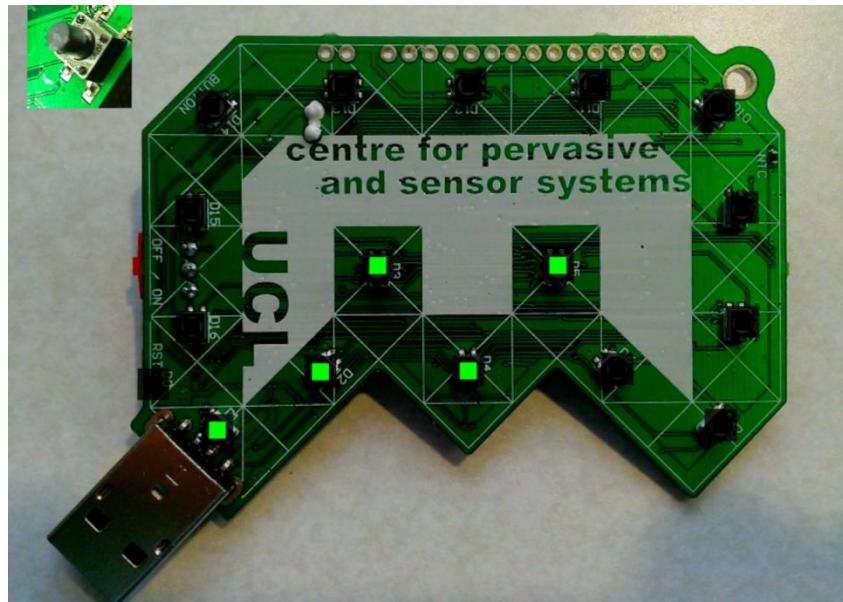
In the code above, we add a conditional statement to check if the user have completed the required amount of steps to light up all the LED on the Engduino. If the user have completed the requirement, the Engduino will start flashing to indicate that he/she is done.

COMPLETING OUR Pedometer!

```
action main ()  
    ▷ setup  
    var counter := 0  
  
    We declare a counter variable to count the number of steps the user has taken in total  
  
    while true do  
        var p := engduino → acceleration  
        var x := p → x  
        var y := p → y  
        var z := p → z  
  
        We assign 3 different variables to store our X,Y and Z values of the Engduino's Accelerometer  
        var acceleration := x * x + y * y + z * z  
        acceleration := math → sqrt(acceleration)  
  
        In order to calculate the acceleration from the Engduino, we have to sum the square of x,y,z and square root  
        the summation  
        if acceleration ≥ 1.1 then  
            counter := counter + 1  
            For us to register a step, we have to make sure the acceleration from the user is at least of acceleration 1.1  
        else do nothing end if  
        ▷ StepCounter(counter)  
        We create a new function to show the number of steps taken  
    end while  
end action
```

In order to finish our Pedometer, we have to add the "StepCounter" function in our main function and pass in the "counter" variable as the input parameter!

Final Product!



CHALLENGE PROBLEM

For every 4 completed sets of steps taken, the LED will light up from Red to Orange to Yellow and lastly Green. The first 4 set of LED should be Red and once the user complete the 5th set, all the lighted LED will change to Orange instead. The table below shows the full problem that you should solve!

Challenge your friends to see who is able to solve this problem the fastest and show the result to your teacher!

Steps taken (Steps needed for one set: 20)	LED color	LED
0 - 4 * Steps needed (0 - 80 steps)	Red	1,2,3,4
5 * Steps needed - 8 * Steps Needed (100 - 160 steps)	Orange	1,2,3,4,5,6,7,8
9 * Steps needed - 12 * Steps needed (180 steps - 240 steps)	Yellow	1,2,3,4,5,6,7,8,9,10,11,12
13 * Steps needed - 16 * Steps needed (260 steps - 320 steps)	Green	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
Above 16 * Steps needed (Above 320 steps)	Green	All LED blinking

REFERENCES

Full working code of tutorial : [TouchDevelop Pedometer](#)

TEACHER NOTES

UPLOADING TOUCHDEVELOP SCRIPT TO ENGDUINO (THROUGH SERIAL PORT)

The following instructions enable you to upload the TouchDevelop scripts onto the Engduino for Windows device. In order to upload the scripts, make sure your Engduino is connected to your device via USB.

1. INSTALL PYTHON 2.7.9 -

<https://www.python.org/downloads/release/python-279/>

Make sure you download the 2.7.9 version, as this is the specific version required for the TouchDevelop to upload the scripts. Run the installer after the file has downloaded.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		5eebcaa0030dc4061156d3429657fb83	16657930	SIG
XZ compressed source tarball	Source release		38d530f7efc373d64a8fb1637e3baaa7	12164712	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	8d8a26fed767302ff38bc5056612c73a	23759976	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	307c2b99a212204e7a1182a354328e94	22006891	SIG
Windows debug information files	Windows		c5838ec1cdd529a7065902c7573d1607	25969730	
Windows debug information files for 64-bit binaries	Windows		544e1137e8ecdce4f4cd2954ea520fa7	23979074	
Windows help file	Windows		dd438e999824c48001e54a2138c4f455	6120616	
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	21ee51a9f44b7160cb6fc68e29a1ddd0	18833408	
Windows x86 MSI installer	Windows		3ed20d8b06dcd339f814b38861f88fc9	18309120	

2. INSTALL NODE.JS [HTTPS://NODEJS.ORG/DOWNLOAD/](https://nodejs.org/download/)

Download the appropriate installer and run the installer for node.js

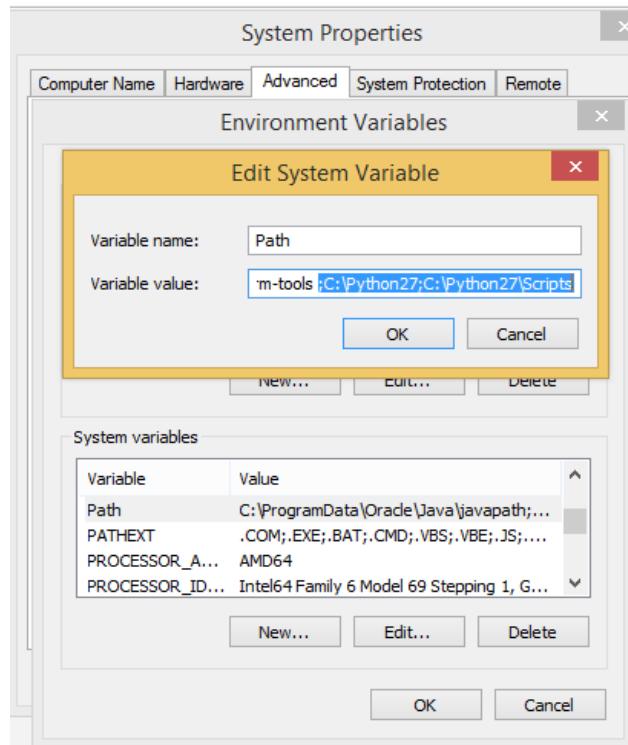


3. ADD PYTHON TO THE PATH VARIABLE

- Go to **Control Panel > All Control Panel Items > System > Advanced System Settings**
- In the **Advanced** tab, click on **Environment Variables**
- Select the **PATH variable > Edit**

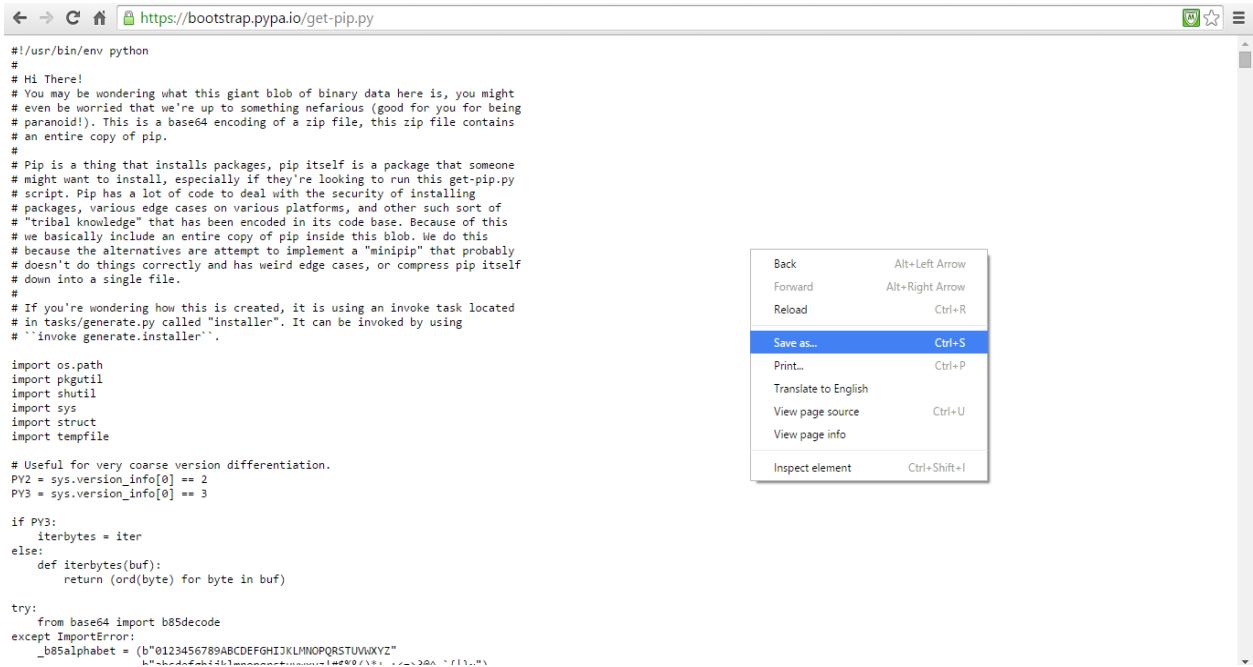
Add **;C:\Python27;C:\Python27\Scripts;** to the existing **Variable value**.

Click **OK** to save the changes.



4. DOWNLOAD GET-PIP.PY [HTTPS://BOOTSTRAP.PYPAA.IO/GET-PIP.PY](https://bootstrap.pypa.io/get-pip.py)

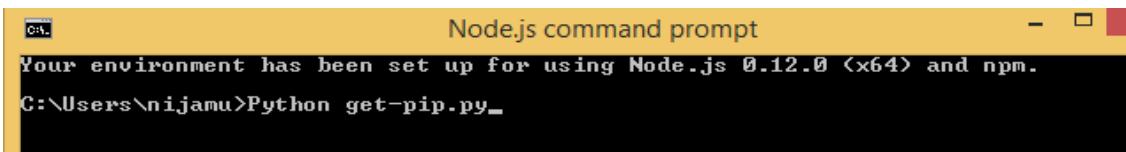
Save the file in a folder that you can easily find.



5. PYTHON GET-PIP.PY

Open node.js command prompt and go to the directory where you saved the file in step 4.

Type *Python get-pip.py* and press enter to run.



6. INSTALL PLATFORMIO

In the command prompt type:

platformio - pip install platformio && pip install --egg scons

or just:

pip install platformio

```
C:\Users\nijamu>pip install platformio && pip install --egg scons
```

7. INSTALL TOUCHDEVELOP LOCAL

Create a folder to save the files for TouchDevelop.

Navigate to the folder in node.js command prompt and type:

npm install -g <http://aka.ms/touchdevelop.tgz>

```
C:\Users\nijamu>cd Desktop
C:\Users\nijamu\Desktop>cd TD
C:\Users\nijamu\Desktop\TD>npm install -g http://aka.ms/touchdevelop.tgz
```

8. RUN TOUCHDEVELOP

To run TouchDevelop local, type:

touchdevelop

```
C:\Users\nijamu>cd Desktop
C:\Users\nijamu\Desktop>cd TD
C:\Users\nijamu\Desktop\TD>touchdevelop
```

BUGS WITH ENGDUINO AND TOUCHDEVELOP

The tutorials that we have included that uses the Engduino are using an Engduino emulator that is included in the TouchDevelop system on the website. This is because there were some bugs when uploading the code onto the Engduino, which meant that all the functions would not work properly.

The problems that came up are listed below:

1. Setting the LED's individually

We found that, although on TouchDevelop, using the emulator, we could set the LED's individually. However, when we uploaded the code onto the Engduino, an error came up regarding 'set_LED'.

```
0000.000> arduino: Oops, the sketch compilation failed. Please review the logs.  
0000.001> shell: error: src.ino: In function 'void StepCounter(int)':  
src.ino:46:36: error: 'set_LED' was not declared in this scope  
scons: *** [.pioenvs\myenv\src\src.o] Error 1  
[ ERROR ] Took 1.20 seconds  
0000.001> shell: [03/26/15 20:55:59] Processing myenv (platform: atmelavr, board: engduinov3, framework: arduino)  
  
avr-g++ -o .pioenvs\myenv\src\src.o -c -fno-exceptions -fno-threadsafe-statics -g -Os -Wall -ffunction-sections -fdata-sections -MMD -mmcu=atmega32u4 -DF_CPU=8000000L -DUSB_VID=0x1B4F -DUSB_PID=0x9208 -DUSB_PRODUCT="\"EngduinoV3\" -DARDUINO=10601 -DPLATFORMIO=010200 -I.pioenvs\myenv\Wire\utility -I.pioenvs\myenv\Wire -I.pioenvs\myenv\EngduinoLEDs -I.pioenvs\myenv\SPI -I.pioenvs\myenv\EngduinoButton -I.pioenvs\myenv\EngduinoAccelerometer -I.pioenvs\myenv\EngduinoThermistor -I.pioenvs\myenv\EngduinoTD -I.pioenvs\myenv\FrameworkArduino -I.pioenvs\myenv\FrameworkArduinoVariant .pioenvs\myenv\src\src.cpp  
0000.002> shell: shell 1  
0000.002> exited with 1  
0000.025> [ ERROR ] Took 1.20 seconds  
0000.058> scons: *** [.pioenvs\myenv\src\src.o] Error 1  
0000.064> src.ino:46:36: error: 'set_LED' was not declared in this scope
```

2. TouchDevelop only uses 'int' values

As our topic was 'acceleration', we wanted to use the accelerometer in our tutorials. These values should be stored as 'double' in C++. However TouchDevelop uploads the value as 'int' onto the Engduino.

```
EngduinoLEDs.begin();  
EngduinoAccelerometer.begin();  
EngduinoButton.begin();  
EngduinoThermistor.begin();  
EngduinoTD.begin();  
  
}  
  
void loop()  
{  
    int totalSteps = 0;  
    TD_Vector3 p = TDLIB_Engduino::acceleration();  
    int x = p.x();  
    int y = p.y();  
    int z = p.z();  
    int steps = (((x*x)+(y*y))+(z*z));  
    steps = sqrt(steps);  
    if ((steps>=1.1)) {  
        totalSteps = (totalSteps+1);  
    }
```

3. The emulator that is on the TouchDevelop system is based on the Engduino v2 (green), whereas we use the Engduino v3 (white). There are small differences between the two models, for example the labeling of the LED's and so take note which version of the Engduino is being used.

NOW YOU'RE GOOD TO GO! ☺