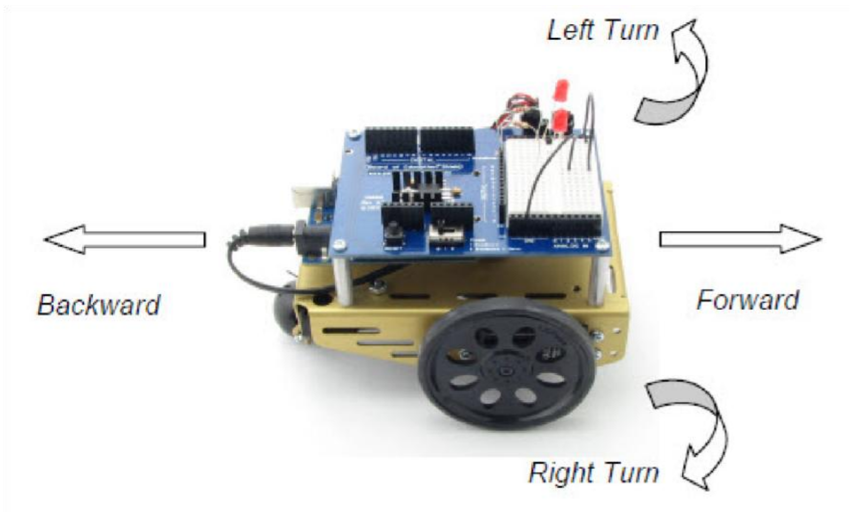
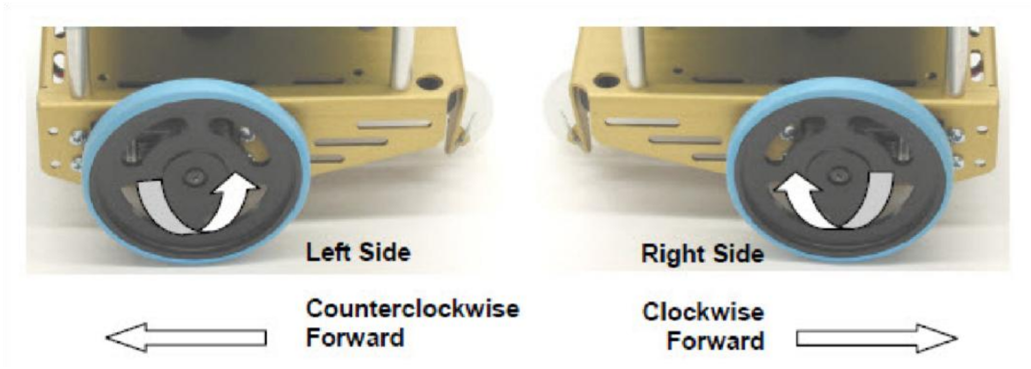
Lab 5

This Lab introduces different programming strategies to make the BOE Shield-Bot move. Once we understand how basic navigation works, we'll make functions for each manoeuvre.



Moving Forward

Have you ever thought about what direction a car's wheels have to turn to propel it forward? The wheels turn opposite directions on opposite sides of the car. Likewise, to make the BOE Shield-Bot go forward, its left wheel has to turn counterclockwise, but its right wheel has to turn clockwise.



Remember that a sketch can use the Servo library's `writeMicroseconds` function to control the speed and direction of each servo. Then, it can use the `delay` function to keep the servos running for certain amounts of time before choosing new speeds and directions. Here's an example that will make the BOE Shield-Bot roll forward for about three seconds, and then stop.

Example Sketch: ForwardThreeSeconds

- ✓ Make sure the BOE Shield's power switch is set to 1 and the battery pack is plugged into the Arduino.
- ✓ Enter, save, and upload ForwardThreeSeconds to the Arduino.
- ✓ Disconnect the programing cable and put the BOE Shield-Bot on the floor.
- ✓

While holding down the Reset button, move the switch to position 3, and then let go. The BOE Shield-Bot should drive forward for three seconds.

```
// Robotics with the BOE Shield - ForwardThreeSeconds
// Make the BOE Shield-Bot roll forward for three seconds, then stop.

#include <Servo.h>                      // Include servo library


Servo servoLeft;                        // Declare left and right servos
Servo servoRight;


void setup()                            // Built-in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                  // Attach right signal to pin 12

  // Full speed forward
servoLeft.writeMicroseconds(1700);      // Left wheel counterclockwise
servoRight.writeMicroseconds(1300);     // Right wheel clockwise
delay(3000);                            // ...for 3 seconds


servoLeft.detach();                          // Stop sending servo signals
servoRight.detach();
}


void loop()                               // Main loop auto-repeats
{                                       // Empty, nothing needs repeating
}
```

How ForwardThreeSeconds Works

First, the Servo library has to be included so that your sketch can access its functions:

```
#include <Servo.h>                      // Include servo library
```

Next, an instance of Servo must be declared and uniquely named for each wheel:

```
Servo servoLeft;                        // Declare left & right servos
Servo servoRight;
```

Instance of an Object

An object is a block of pre-written code that can be copied and re-used multiple times in a single sketch. Each copy, called an object instance, can be configured differently. For example, the two `Servo` declarations create two instances of the object's code, named `servoLeft` and `servoRight`. Then, functions within each instance can be called and configured individually. So, `servoLeft.attach(13)` configures the `servoLeft` object instance to send its servo control signals to pin 13. Likewise, `servoRight.attach(12)` tells the `servoRight` object instance to send its signals to pin 12.

A sketch automatically starts in its `setup` function. It runs the code in there once before moving on to the `loop` function, which automatically keeps repeating. Since we only want the BOE Shield-Bot to go forward and stop once, all the code can be placed in the `setup` function. This leaves the `loop` function empty, but that's okay.

As with all motion sketches, the first action `setup` takes is making the piezospeaker beep. The tone function call transmits a signal to digital pin 4 that makes the piezospeaker play a 3 kHz tone that lasts for 1 second. Since the tone function works in the background while the code moves on, delay(1000) prevents the BOE Shield-Bot from moving until the tone is done playing.

```
void setup()                              // Built-in initialization
  {
    tone(4, 3000, 1000);              // Play tone for 1 second
    delay(1000);                          // Delay to finish tone
```

Next, the `servoLeft` object instance gets attached to digital pin 13 and the `servoRight` instance gets attached to pin 12. This makes calls to `servoLeft.writeMicroseconds` affect the servo control signals sent on pin 13.
Likewise, calls to `servoRight.writeMicroseconds` will affect the signals sent on pin 12.

```
servoLeft.attach(13);                // Attach left signal to pin 13
servoRight.attach(12);               // Attach right signal to pin 12
```

Remember that we need the BOE Shield-Bot's left and right wheels to turn in opposite directions to drive forward. The function call `servoLeft.writeMicroseconds(1700)` makes the left servo turn full speed counterclockwise, and the function call `servoRight.writeMicroseconds(1300)` makes the right wheel turn full speed clockwise. The result is forward motion. The `delay(3000)` function call keeps the servos running at that speed for three full seconds. After the delay, `servoLeft.detach` and `servoRight.detach` discontinue the servo signals, which bring the robot to a stop.

```
// Full speed forward
servoLeft.writeMicroseconds(1700);  // Left wheel counterclockwise
servoRight.writeMicroseconds(1300);
// Right wheel clockwise
delay(3000);                         // ...for 3 seconds
```

```
servoLeft.detach();                  // Stop sending servo signals
servoRight.detach();
}
```

After the `setup` function runs out of code, the sketch automatically advances to the `loop` function, which repeats itself indefinitely. In this case, we are leaving it empty because the sketch is done, so it repeats nothing, over and over again, indefinitely.

```
void loop()                     // Main loop auto-repeats
{
}
```

Your Turn – Adjusting Distance

Want to change the distance traveled? Just change the time in delay(3000). For example, delay(1500) will make the BOE Shield-Bot go for only half the time, which in turn will make it travel only half as far. Likewise, delay(6000) will make it go for twice the time, and therefore twice the distance.

✓ Change `delay(3000)` to `delay(1500)` and re-upload the sketch. Did the BOE Shield-Bot travel only half the distance?

Moving Backward, Rotating, and Pivoting

All it takes to get other motions out of your BOE Shield-Bot are different combinations of `us` parameters in your `servoLeft` and `servoRight writeMicroseconds` calls. For example, these two calls will make your BOE Shield-Bot go backwards:

```
 // Full speed backwards
 servoLeft.writeMicroseconds(1300);   // Left wheel clockwise
servoRight.writeMicroseconds(1700);  // Right wheel counterclockwise
```

These two calls will make your BOE Shield-Bot rotate in place to make a left turn:

```
 // Turn left in place
 servoLeft.writeMicroseconds(1300);   // Left wheel clockwise
servoRight.writeMicroseconds(1300);  // Right wheel clockwise
```

These two calls will make your BOE Shield-Bot rotate in place for a right turn:

```
 // Turn right in place
 servoLeft.writeMicroseconds(1700);   // Left wheel counterclockwise
servoRight.write Microseconds(1700); // Right wheel counterclockwise
```

Let's combine all these commands into a single sketch that makes the BOE Shield-Bot move forward, turn left, turn right, then move backward.

Example Sketch: ForwardLeftRightBackward

```
// Robotics with the BOE Shield - ForwardLeftRightBackward
// Move forward, left, right, then backward for testing and tuning.
```

```
#include <Servo.h>                           // Include servo library

Servo servoLeft;                        // Declare left and right servos
Servo servoRight;


void setup()                            // Built-in initialization block
{
tone(4, 3000, 1000);                       // Play tone for 1 second
delay(1000);                               // Delay to finish tone

servoLeft.attach(13);                   // Attach left signal to pin 13
servoRight.attach(12);                    // Attach right signal to pin 12

// Full speed forward
```

```
 servoLeft.writeMicroseconds(1700);    // Left wheel counterclockwise
 servoRight.writeMicroseconds(1300);        // Right wheel clockwise
 delay(2000);                               // ...for 2 seconds

 // Turn left in place
 servoLeft.writeMicroseconds(1300);         // Left wheel clockwise
 servoRight.writeMicroseconds(1300);        // Right wheel clockwise
 delay(600);                                // ...for 0.6 seconds

 // Turn right in place
 servoLeft.writeMicroseconds(1700);         // Left wheel counterclockwise
 servoRight.writeMicroseconds(1700);       // Right wheel counterclockwise
 delay(600);                                // ...for 0.6 seconds

 // Full speed backward
 servoLeft.writeMicroseconds(1300);         // Left wheel clockwise
 servoRight.writeMicroseconds(1700);       // Right wheel counterclockwise
 delay(2000);                               // ...for 2 seconds

 servoLeft.detach();                        // Stop sending servo signals
 servoRight.detach();
 }

 void loop()                                 // Main loop auto-repeats
 {                                   // Empty, nothing needs repeating
 }
```

Your Turn – Pivoting

You can make the BOE Shield-Bot turn by pivoting around one wheel. The trick is to keep one wheel still while the other rotates. Here are the four routines for forward and backward pivot turns:

```
// Pivot forward-left
 servoLeft.writeMicroseconds(1500);   // Left wheel stop
servoRight.writeMicroseconds(1300);  // Right wheel clockwise

 // Pivot forward-right
 servoLeft.writeMicroseconds(1700);   // Left wheel counterclockwise
servoRight.writeMicroseconds(1500);  // Right wheel stop
```

```
 // Pivot backward-left
 servoLeft.writeMicroseconds(1500);   // Left wheel stop
 servoRight.writeMicroseconds(1700);  // Right wheel counterclockwise

 // Pivot backward-right
 servoLeft.writeMicroseconds(1300);     // Left wheel clockwise
servoRight.writeMicroseconds(1500);    // Right wheel stop
```

Imagine writing a sketch that instructs your BOE Shield-Bot to travel full-speed forward for fifteen seconds. What if your robot curves slightly to the left or right during its travel, when it's supposed to be traveling straight ahead? There's no need to take the BOE Shield-Bot back apart and re-adjust the servos with a screwdriver to fix this. You can simply adjust the sketch slightly to get both wheels traveling the same speed. While the screwdriver approach could be considered a hardware adjustment, the programming approach would be a software adjustment.

Straightening the Shield-Bot's Path

So, would your BOE Shield-Bot travel in an arc instead of in a straight line? Top speed varies from one servo to the next, so one wheel is likely to rotate a little faster than the other, causing the BOE Shield-Bot to make a gradual turn.

To correct this, the first step is to examine your BOE Shield-Bot's forward travel for a longer period of time to see if it is curving, and which way, and how much.

```
// Robotics with the BOE Shield - ForwardTenSeconds
  // Make the BOE Shield-Bot roll forward for ten seconds, then stop.

  #include <Servo.h>          // Include servo library


  Servo servoLeft;             // Declare left and right servos
  Servo servoRight;


  void setup()                 // Built-in initialization block
  {
    tone(4, 3000, 1000);       // Play tone for 1 second
```

```
    delay(1000);                 //   Delay to finish tone

  servoLeft.attach(13);         // Attach left signal to pin 13
  servoRight.attach(12);        // Attach right signal to pin 12

    // Full speed forward
  servoLeft.writeMicroseconds(1700);        // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300);        // Right wheel clockwise
  delay(10000);                              // ...for 10 seconds

  servoLeft.detach();                        // Stop sending servo signals
  servoRight.detach();
  }


  void loop()                               // Main loop auto-repeats
  {                                  // Empty, nothing needs repeating
  }
```

Your Turn – Adjusting Servo Speed to Straighten the BOE Shield-Bot's Path

If your BOE Shield-Bot turns slightly when you want it to go straight forward, the solution is fairly simple. Just slow down the faster wheel. Remember from the servo transfer curve graph that you have best speed control over the servos in the 1400 to 1600 µs range.

us Parameters in writeMicroseconds(us)

Top speed clockwise	Linear speed zone starts	Full stop	Linear speed zone ends	Top speed counterclockwise
1300	1400	1500	1600	1700

Let's say that your BOE Shield-Bot gradually turns left. That means the right wheel is turning faster than the left. Since the left wheel is already going as fast as it possibly can, the right wheel needs to be slowed down to straighten out the robot's path. To slow it down, change the `us` parameter in `servoRight.writeMicroseconds(us)` to a value closer to 1500. First, try 1400. Is it still going too fast? Raise it 1410. Keep raising the parameter by 10 until the BOE Shield-Bot no longer curves to the left. If any adjustment overshoots 'straight' and your BOE Shield-Bot starts curving to the right instead, start decreasing the us parameter by smaller amounts. Keep refining that `us` parameter until your BOE ShieldBot goes straight forward. This is called an iterative process, meaning that it takes repeated tries and refinements to get to the right value.

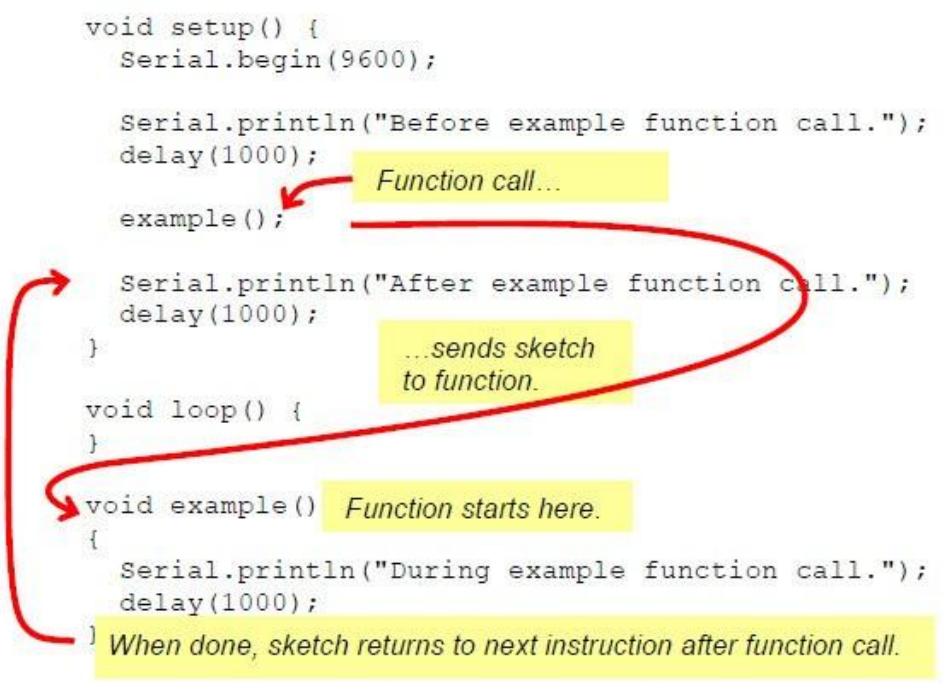
Simplify Navigation with Functions

One convenient way to execute pre-programmed manoeuvres is with functions. In the next chapter, your BOE Shield-Bot will have to perform manoeuvres to avoid obstacles, and a key ingredient for avoiding obstacles is executing pre-programmed manoeuvres.

The `setup` and loop functions are built into the Arduino language, but you can add more functions that do specific tasks for your sketch. This activity introduces how to add more functions to your sketch as well as a few different approaches to creating reusable manoeuvres with those functions.

Minimal Function Call

The diagram below shows part of a sketch that contains a function named `example` added at the end, below `the` loop function. It begins and gets named with the function definition `void example()`. The empty parentheses means that it doesn't need any parameters to do its job, and `void` indicates that it does not return a value (we'll look at functions that return values in a later chapter). The curly braces `{}` that follow this definition contain the `example` function's block of code.

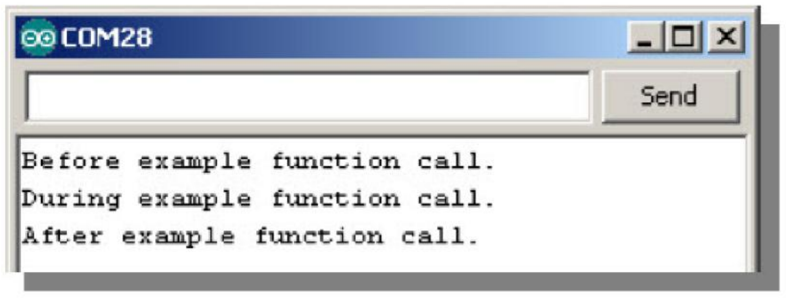


```
void setup() {
  Serial.begin(9600);

  Serial.println("Before example function call.");
  delay(1000);

  example();

  Serial.println("After example function call.");
  delay(1000);
}

void loop() {
}

void example()
{
  Serial.println("During example function call.");
  delay(1000);
}
```

Function call…
…sends sketch to function.
Function starts here.
When done, sketch returns to next instruction after function call.

There is a function call to example in the setup function, labeled in the diagram above. That example() line tells the sketch to go find the function with that name, execute its code, and come back when done. So, the sketch jumps down to void example() and executes the two commands in its curly braces. Then, it returns to the function call and continues from there. Here is the order of events you will see when you run the sketch:

1. The Serial Monitor displays "Before example function call."
2. After a one second delay, the monitor displays "During example function call." Why? Because the `example()` call sends the code to `void example()`, which has the line of code that prints that message, followed by a 1 second delay. Then, the function returns to the `example` call in `setup`.
3. The Serial Monitor displays "After example function call."



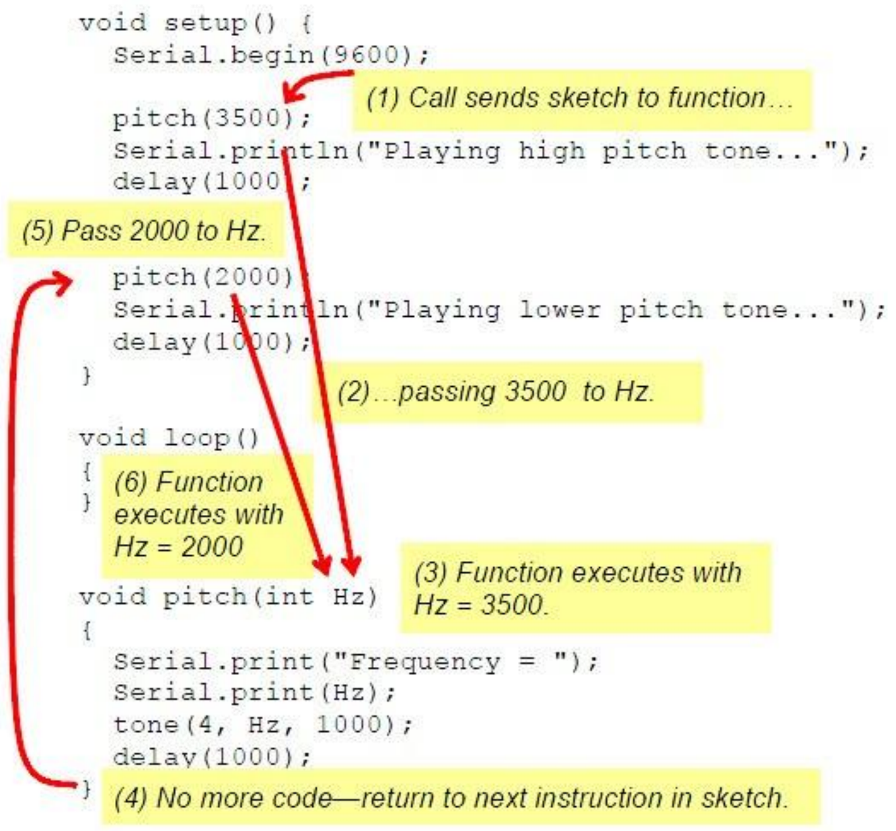
```
Before example function call.
During example function call.
After example function call.
```

```
//Robotics with the BOE Simple Function Call

void setup() {
Serial.begin(9600);


Serial.println("Before example function call.");
delay(1000);   example();        // This is the function call

Serial.println("After example function call.");
delay(1000);
}

void loop() {
}

void example()                              // This is the function
{
   Serial.println("During example function call.");   delay(1000); }
```

Function Call with Parameters

Remember that a function can have one or more parameters—data that the function receives and uses when it is called. The diagram below shows the `pitch` function from the next sketch. It is declared with `void pitch(int Hz)`. Recall from and earlier Lab that the Arduino stores variable values in different data types, with `int` specifying an integer value in the range of -32,768 to 32,767. Here, the term `int Hz` in the parentheses defines a parameter for the `pitch` function; in this case, it declares a local variable `Hz` of data type `int`.
Local variables, remember, are declared within a function, and can only be seen and used inside that function. If a local variable is created as a parameter in the function declaration, as void pitch(int Hz) is here, initialize it by passing a value to it each time the function is called. For example, the call `pitch(3500)` passes the integer value 3500 to the `pitch` function's `int Hz` parameter.

So, when the first function call to `pitch` is made with `pitch(3500)`, the integer value 3500 gets passed to `Hz`. This initializes `Hz` to the value 3500, to be used during this trip through the `pitch` function's code block. The second call, `pitch(2000)`, initializes `Hz` to 2000 during the sketch's second trip through the `pitch` function's code block.


(1) Call sends sketch to function…
(2)…passing 3500 to Hz.
(3) Function executes with Hz = 3500.
(4) No more code—return to next instruction in sketch.
(5) Pass 2000 to Hz.
(6) Function executes with Hz = 2000

```
void setup() {
  Serial.begin(9600);

  pitch(3500);
  Serial.println("Playing high pitch tone...");
  delay(1000);

  pitch(2000);
  Serial.println("Playing lower pitch tone...");
  delay(1000);
}

void loop()
{
}

void pitch(int Hz)
{
  Serial.print("Frequency = ");
  Serial.print(Hz);
  tone(4, Hz, 1000);
  delay(1000);
}
```

```
// This program demonstrates a function call with a
parameter.

void setup() {   Serial.begin(9600);

Serial.println("Playing higher pitch tone...");   pitch(3500);

// pitch function call passes 3500 to Hz parameter

delay(1000);


Serial.println("Playing lower pitch tone...");   pitch(2000);

// pitch function call passes 2000 to Hz parameter


  delay(1000);
}
```

```
void loop()
{
}

void pitch(int Hz)     // pitch function with Hz declared as a parameter
{
  Serial.print("Frequency = ");
Serial.println(Hz)
; tone(4, Hz, ms);
delay(ms);
}
```

Put Maneuvers Into Functions

Let's try putting the `forward`, `turnLeft`, `turnRight`, and `backward` navigation routines inside functions. Here's an example:

Example Sketch – MovementsWithSimpleFunctions

Enter, save, and upload MovementsWithSimpleFunctions.

```
  // Move forward, left, right, then backward for testing and tuning.

#include <Servo.h>        // Include servo library
```

```
Servo servoLeft;          // Declare left and right servos
Servo servoRight;


void setup()              // Built-in initialization block
{
  tone(4, 3000, 1000);                          // Play tone for 1 second
  delay(1000);                                  // Delay to finish tone

servoLeft.attach(13);                        // Attach left signal to pin 13
servoRight.attach(12);                        // Attach right signal to pin 12

forward(2000);                                // Go forward for 2 seconds
turnLeft(600);                                // Turn left for 0.6 seconds
turnRight(600);                               // Turn right for 0.6 seconds
backward(2000);                               // go backward for 2 seconds

  disableServos();                              // Stay still indefinitely
}


void loop()                          // Main loop auto-repeats
{                                      // Empty, nothing needs repeating
}

void forward(int time)                       // Forward function
{
servoLeft.writeMicroseconds(1700);          // Left wheel counterclockwise
servoRight.writeMicroseconds(1300);         // Right wheel clockwise
delay(time);                                // Maneuver for time ms
}

void turnLeft(int time)                         // Left turn function
{
  servoLeft.writeMicroseconds(1300);            // Left wheel clockwise
servoRight.writeMicroseconds(1300);         // Right wheel clockwise
delay(time);                                // Maneuver for time ms
}

void turnRight(int time)                        // Right turn function
{
```

```
servoLeft.writeMicroseconds(1700);         // Left wheel counterclockwise
servoRight.writeMicroseconds(1700);        // Right wheel counterclockwise
delay(time);                               // Maneuver for time ms
}

void backward(int time)                      // Backward function
{
servoLeft.writeMicroseconds(1300);         // Left wheel clockwise
servoRight.writeMicroseconds(1700);        // Right wheel counterclockwise
delay(time);                               // Maneuver for time ms
}

void disableServos()                         // Halt servo signals
{
servoLeft.detach();                        // Stop sending servo signals
servoRight.detach();
}
```

You should recognize the pattern of movement your BOE Shield-Bot makes; it is the same one made by the ForwardLeftRightBackward sketch. This is a second example of the many different ways to structure a sketch that will result in the same movements.

Your Turn – Move Function Calls into loop

Want to keep performing that set of four maneuvers over and over again? Just move those four maneuvering function calls from the `setup` function into the `loop` function. Try this:

Save the sketch under a new name, like MovementsWithFunctionsInLoop

Comment out the `disableServos()` function call that's in `setup` by placing two forward slashes to its left, like this: `// disableServos`

Remove the `// Empty…` comment from the `loop` function—it won't be correct!

Cut the function calls to `forward(2000)`, `turnLeft(600)`, `turnRight(600)`, and `backward(2000)` out of the `setup` function and paste them into the `loop` function. It should look like this when you're done:

```
void setup()                         // Built-in initialization block
{
  tone(4, 3000, 1000);               // Play tone for 1 second
delay(1000);                       // Delay to finish tone

  servoLeft.attach(13);              // Attach left signal to pin 13
servoRight.attach(12);             // Attach right signal to pin 12

  // disableServos();                  // Stay still indefinitely
}


void loop()                          // Main loop auto-repeats
{
forward(2000);                     // Go forward for 2 seconds
turnLeft(600);                     // Turn left for 0.6 seconds
turnRight(600);                    // Turn right for 0.6 seconds
backward(2000);                    // go backward for 2 seconds
```

Upload the modified sketch and verify that it repeats the sequence of four maneuvers indefinitely.

Custom Manoeuvre Function

The last sketch, MovementsWithSimpleFunctions, was kind of long and clunky. And, the four functions it uses to drive the robot are almost the same. The TestManeuverFunction sketch takes advantage of those function's similarities and streamlines the code.

TestManeuverFunction has a single function for motion named `maneuver` that accepts three parameters: `speedLeft`, `speedRight`, and `msTime`:

```
void maneuver(int speedLeft, int speedRight, int msTime)
```

The rules for `speedLeft` and `speedRight,` listed below, are easy to remember. Best of all, with this `maneuver` function you don't have to think about clockwise and counterclockwise rotation anymore.

positive values for moving the robot
- forward negative values for
- moving the robot backward
- 200 for full speed forward
- –200 for full speed backward
- 0 for stop
- 100 to –100 range for linear speed control

The rules for `msTime` are:

```
maneuver(200, 200, 2000);          // Forward 2 seconds
maneuver(-200, 200, 600);          // Left 0.6 seconds
maneuver(200, -200, 600);          // Right 0.6 seconds
maneuver(-200, -200, 2000);        // Backward 2 seconds
maneuver(0, 0, -1);                // Disable servos
```

```
 // Move forward, left, right, then backward with maneuver function.

 #include <Servo.h>                           // Include servo library


 Servo servoLeft;                       // Declare left and right servos
 Servo servoRight;


 void setup()                             // Built-in initialization block
 {
   tone(4, 3000, 1000);                       // Play tone for 1 second
 delay(1000);                               // Delay to finish tone

   servoLeft.attach(13);            // Attach left signal to pin 13
 servoRight.attach(12);             // Attach right signal to pin 12


 maneuver(200, 200, 2000);                    // Forward 2 seconds
 maneuver(-200, 200, 600);                    // Left 0.6 seconds
 maneuver(200, -200, 600);                    // Right 0.6 seconds
 maneuver(-200, -200, 2000);                  // Backward 2 seconds
 maneuver(0, 0, -1);                          // Disable servos
```

```
}


void loop()                                  // Main loop auto-
repeats {                          // Empty, nothing needs repeating
}

void maneuver(int speedLeft, int speedRight, int msTime)
{
// speedLeft, speedRight ranges: Backward Linear Stop Linear   Forward
// -200      -100......0......100       200
servoLeft.writeMicroseconds(1500 + speedLeft);
// Set Left servo speed
servoRight.writeMicroseconds(1500 - speedRight);
// Set right servo speed
if(msTime==-1)                               // if msTime = -1
  {
servoLeft.detach();                           // Stop servo signals
servoRight.detach();
  }
  delay(msTime);                              // Delay for msTime
                                                                      }6
```

- Positive values for the number of ms to execute the maneuver
- –1 to disable the servo signal