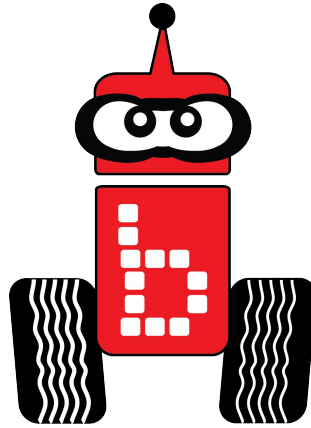




Functions with Arguments

- **Key Concepts**
 -
- **Pacing**
 -

Table Of Contents

[Discovering Functions with Arguments-Activity 1](#)

Write your own Function with Arguments

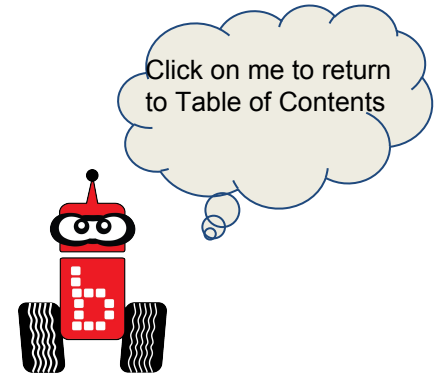
[Activity 2](#)

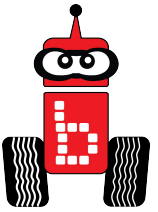
[Activity 3](#)

[Activity 4](#)

[A Function within a Function-Activity 5](#)

[Assessments and Rubrics](#)





Standards

Goal:

- Students will familiarize themselves with the functions `msleep()` and `motor()`
- Students will understand how to move their robots in the following manner: forwards, backwards, straight, circles, right and left turns

Standards:

Common Core State Standards Math Practices

CCSSMP1: Make sense of problems and persevere in solving them

CCSSMP2: Reason abstractly and quantitatively

CCSSMP4: Model with mathematics

CCSSMP6: Attend to precision

CCSSMP8: Look for and express regularity in repeated reasoning

Next Generation Science and Engineering Practice

1: Asking questions and defining problems

2: Developing and using models

3: Planning and carrying out investigations

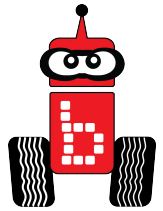
4: Analyzing and interpreting data

5: Using mathematics and computational thinking

6: Constructing explanations and designing solution

7: Engaging in argument from evidence obtaining, evaluating, and communicating information

Standards Continued



Standards Continued:

2016 ISTE Standards

Empowered Learner

1c: Students use technology to seek feedback that informs and improves their practice and to demonstrate their learning in a variety of ways.

1d: Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

Knowledge Constructor

3d: Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Innovative Designer

4a: Students know and use a deliberate design process for generating ideas, testing theories, creating innovative artifacts or solving authentic problems.

4b: Students select and use digital tools to plan and manage a design process that considers design constraints and calculated risks.

4c: Students develop, test and refine prototypes as part of a cyclical design process.

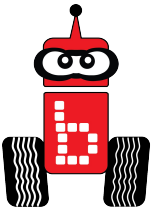
4d: Students exhibit a tolerance for ambiguity, perseverance and the capacity to work with open-ended problems.

Computational Thinker

5a: Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions.

Discovering Functions with Arguments:

Activity 1

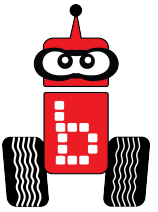


1. Do a KWL chart to discuss the following slides

Parts of a Function Review

Function Argument

Writing Your Own Function with an Argument

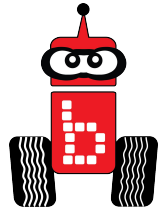


Parts of a Function Review

From [Unit 12](#) we are familiar with 3 components of a function and the order in which they should be created :

1. Function prototype
 - a. Create a **function prototype** on the line *after* `#include <kipr/botball.h>` and *before* `int main()`.
2. *Function definition*
 - a. *Define* the new function (i.e., specify what the robot will actually do). The **function definition** goes after the last “}” in your program.
3. *Function Call*
 - a. *Call* (use) the new function by typing the **function name** within a block of code.

Parts of a Function Review



Function prototypes
go above main.

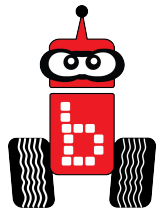
Function calls
go inside main
(or inside other
functions).

Function definitions
go below main.

```
void drive_forward(int milliseconds); // function prototype
```

```
int main()
{
    drive_forward(4000); // function call
    return 0;
} // end main
```

```
void drive_forward(int milliseconds) // function definition
{
    motor(0, 100);
    motor(3, 100);
    msleep(milliseconds);
    ao();
} // end drive_forward
```



Parts of a Function Review

The **function prototype** and the **function definition** look the same except for one thing...

→ `void drive_forward(int milliseconds); // function prototype`

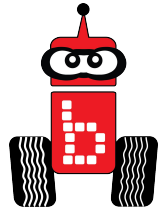
```
int main()
{
    drive_forward(4000); // function call
    return 0;
} // end main
```

→ `void drive_forward(int milliseconds) // function definition`

```
{
    motor(0, 100);
    motor(3, 100);
    msleep(milliseconds);
    ao();
} // end drive_forward
```

Notice: no semicolon!
(Why not?)

Function Arguments



The fourth component of a function is an argument.

4. Function arguments:

a. Values you will set when you call the function

```
1 → void drive_forward(int milliseconds); // function prototype

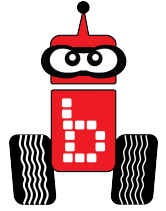
int main()
{
3 → drive_forward(4000); // function call
    return 0;
} // end main

2 → { void drive_forward(int milliseconds) // function definition
    {
        motor(0, 80);
        motor(3, 80);
        msleep(milliseconds);
        ao();
    } // end drive_forward
}
```

Diagram illustrating function arguments with annotations:

- 1: Points to the function prototype `void drive_forward(int milliseconds);`.
- 2: Points to the function definition `void drive_forward(int milliseconds) { ... }`.
- 3: Points to the function call `drive_forward(4000);`.
- 4: Points to the argument `4000` in the function call.

Writing Your Own Functions With Arguments



1. Function Prototype

```
void drive_forward(int milliseconds); // function prototype
```

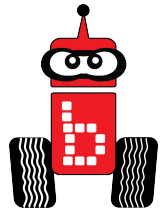
The function argument goes in between the parentheses in the **function prototype**. `int` is used because an integer (milliseconds) is being used.

You can think of a function with an argument as an f of x statement, $f(x)$, statement.

f is the function name `drive_forward`
 (x) is the argument or integer value (`int milliseconds`)

Math Extension

Writing Your Own Functions With Arguments

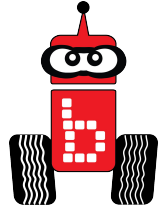


2. Function Call

```
int main()
{
    drive_forward(4000); // function call
    return 0;
} // end main
```

(4000) is the integer value the argument is referring to. Remember the (4000) is the msleep time in milliseconds.

Writing Your Own Functions With Arguments



3. Function Definition

```
void drive_forward(int milliseconds) // function definition
{
    motor(0, 100);
    motor(3, 100);
    msleep(milliseconds);
    ao();
} // end drive_forward
```

- Remember the **function definition** is the same as the **function prototype**, but does not have a semicolon, and it goes at the bottom of the program after the last curly brace.

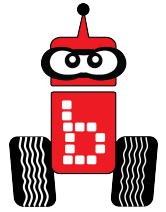
```
void drive_forward(int milliseconds) // function definition
{
    motor(0, 100);
    motor(3, 100);
    msleep(milliseconds);
    ao();
} // end drive_forward
```



- What is inside the curly braces{} is the definition of the **function prototype**.

Write your own Function with Arguments

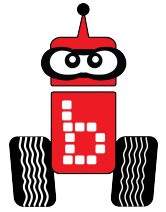
Activity 2



1. Open a new project and copy [slide 7](#).
2. Run your program. What happened?
3. Now, change the argument value (4000) to a value other than 4000.
4. Run your program. Discuss with a partner what changes you observed.

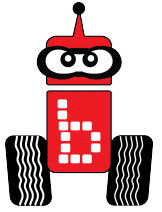
Write your own Function with Arguments

Activity 3



1. Write a new program, in which you create a new function to drive backward, with the function argument of time.
2. You should use [code planning paper](#) to write out the pseudocode or create flowchart.
3. Compile and run your program.
4. Discuss how efficient writing functions and arguments are with a partner.

Writing a Function with Multiple: Arguments-Activity 4



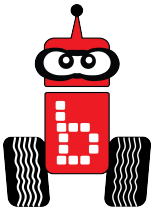
How many arguments are in this function?

```
motor(0, 100);
```

Proceed to the next slide

Writing a Function with Multiple Arguments:

Activity 4- Continued



How many arguments are in this function?

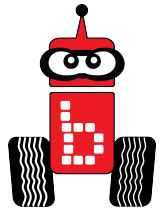
```
motor (0 , 100) ;
```



2 Arguments

Proceed to the next slide

Write a Function with Multiple Arguments: Activity 4- Continued



```
#include <kipr/botball.h>

int drive(int l_motor_speed, int r_motor_speed); // function prototypes

int main()
{
    drive(100,100); // function call
    msleep(4000);
    ao();

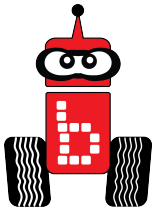
    return 0;
}

int drive(int l_motor_speed, int r_motor_speed) // function definition
{
    motor(0, l_motor_speed);
    motor(3, r_motor_speed);
}
```

Definition of function

Writing a Function with Multiple Arguments:

Activity 4- Continued



What would a function to make your robot follow the path of a Curvy line look like?

- Refer to [Activity 7](#) in Unit 12
- What arguments would be needed to draw a curvy line?
- Use the [code planning paper](#) to help you determine this.



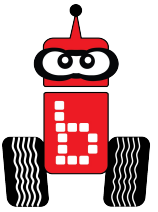
Arguments

1.

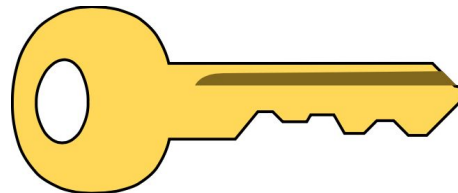
2.

Writing a Function with Multiple Arguments-

Activity 4- Continued



1. Start a new program called “Curvy line”.
2. Write a program using a function with the 2 arguments you created from the previous slide.
3. Run your program.
4. What happened? Use the strategy “[Elbow Partners](#)”.



Activity 4 Solution

```
#include <kipr/botball.h>

int drive(int l_motor_speed, int r_motor_speed); // function prototypes

int main()
{
    drive(30,100); // function call
    msleep(2000);

    drive(100,30); // function call
    msleep(2000);

    drive(30,100); // function call
    msleep(2000);

    drive(100,30); // function call
    msleep(2000);

    ao();

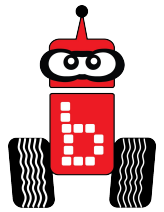
    return 0;
}

int drive(int l_motor_speed, int r_motor_speed) // function definitions
{
    motor(0, l_motor_speed);
    motor(3, r_motor_speed);
}
```

A Function within a Function

Activity 5

1. Read and discuss this slide to understand how to create a function within a function.
 - You can actually use functions you have written inside other functions.
 - For example, you just wrote a function with two arguments:
`l_motor_speed` and `r_motor_speed`.
 - What if you also wanted to provide a function argument to control `msleep`?
 - For example, let's write a `drive_for` function that uses *your* `drive` function to move for a certain amount of time...
2. Proceed to the next slide to see an example.



A Function within a Function:

Activity 5

```
#include <kipr/botball.h>

int drive(int l_motor_speed, int r_motor_speed); // function prototypes
int drive_for(int l_motor_speed, int r_motor_speed, int milliseconds);

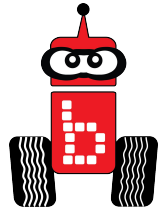
int main()
{
    drive_for(100,100,2000); // function call to drive the robot full speed forward for 2 seconds

    return 0;
}

int drive(int l_motor_speed, int r_motor_speed) // function definitions
{
    motor(0, l_motor_speed);
    motor(3, r_motor_speed);
}

int drive_for(int l_motor_speed, int r_motor_speed, int milliseconds)
{
    drive(l_motor_speed, r_motor_speed);
    msleep(milliseconds);
    ao();
}
```

Assessments and Rubrics



Suggestions: *Understanding* rubric
and or *Group Collaboration*