

# Lesson Plan

Lesson 1: Learning Sequence

Lesson 2: Learning Branch, Jump ( ), goto ( )

Lesson 3: Making decisions, Conditional – if ( )  
then else ( )

**Lesson 4: Fixing Errors, Bug and Debugging**

Lesson 5: Looping with repeat, bounded loops

Lesson 6: Understanding Functions

## *Bonus lessons*

Lesson A: Introducing operations, greater, less  
than. Boolean - TRUE, FALSE

Lesson B: Introducing Variable, string and numeric

Lesson C: Nested repeat - Loopy Loop

## **Lesson 4: Debugging**

**Before you start** – If you haven't already, please read CoderBunnyz Rulebook page 1-8 to be introduced to playing the game.

### **Lesson Overview**

Students will do an introductory worksheet. Then they will play the game level with fences and puddles and will be introduced to debugging with the bug-fix-it. Finally, they will write their code as an algorithm.

### **Lesson Objective**

- First students will do a worksheet to introduce them to debugging. They will understand the concept of bugs in the code and how to debug.
- Then while playing a level of game students will:
  1. Predict the wrong move in the code
  2. Fix the move to find the bug and solve errors
- Finally they will record the steps on the sheet as the algorithm used.

### **Materials needed**

- Debugging worksheet (on the next page), game, pencil, optional - algorithm sheet (4.1)

### **Getting Started**

- Ask the students about a time in their everyday lives in which they had a problem. E.g. - took a wrong turn to school, picked a wrong book from library, etc. How did they fix the problem when something is not working?
- Instructor explains the worksheet and students do the exercise.
- After the worksheet is complete, arrange the game by picking up a maze from RuleBook page 24. Explain the cards, movements

and directions to the players. Choose the destination and get ready to start the game.

### Activity

- Play the level 1.3 of CoderBunnyz to program the bunny to move through the maze, eat its colored carrot and reach the destination.
- Remind the players about using the bugfixit if they want to change or debug their card(s).
- If no bugfixit is used, congratulate the player(s) on writing a bug-free code. *(Optional) write algorithm on 4.1 and bug information*
- A bug free code is the first step to writing efficient codes!

### Fun Fact

Did you know first computer bug was named due to a real bug ?

Grace Hopper recorded the first computer 'bug' in the book as she was working for the MARK II computer.

# 4. Debugging

The process of finding and fixing an incorrect solution (bug)

Read aloud

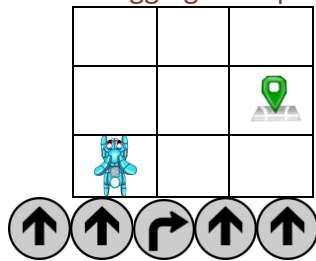


## Debugging a problem

- Look at the error in the current solution
- Think of how to correct it (debug)
- Find and test the correct solution (fix bug)

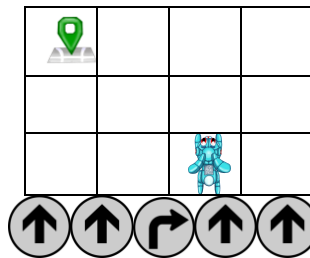
Read aloud

Code debugging example - The second move forward is extra!



Practice Exercise

Your turn to help the bunny now. Find the bug in the code!



Practice Exercise

Find 3 case where you did something wrong and then corrected it (bug-debug)

- 1)
- 2)
- 3)

Practice Exercise





## 4.1 Debug



Look at your code : Count how many code-cards you use in your coding today:

\_\_\_\_\_ MoveForward ( );

\_\_\_\_\_ TurnLeft ( );

\_\_\_\_\_ TurnRight ( );

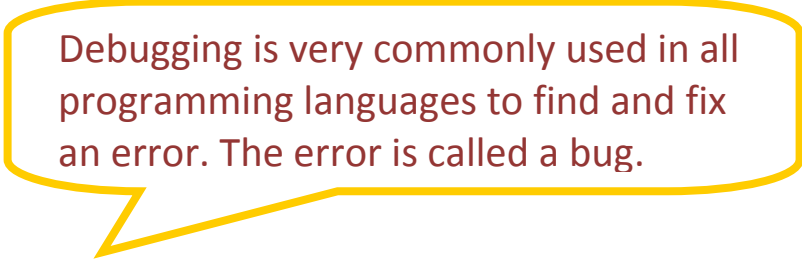
\_\_\_\_\_ Jump ( );



**Using the symbol or text. Write your Code for the game played!!**

**(Bonus – Write how many times you used bugfixit and explain – use back page)**

## How is debugging used in real programming languages?



Debugging is very commonly used in all programming languages to find and fix an error. The error is called a bug.

Debugging can range from fixing simple errors (like some did while playing the game) to performing very long and tiresome tasks of collecting information and reviewing those to find an issue.

There are some software tools available that helps in debugging the program. These tools are called debuggers, and they help the programmer detect errors so they can quickly debug it.

Some programming languages, such as Java, have features that can easily connect various lines of code to detect where the problem is and why it is there. In programming languages such as C or assembly, bugs may cause silent problems such as incorrect data written in the memory, and it is often difficult to see where the problem started out. In those cases, memory debugger tools may be needed.