Rubber Duck Debugging

As a software developer, you’ve probably been in a situation where your code is not working as expected.

This could happen because you made a typo, you weren't paying enough attention, or you weren't sure how something worked.

When your code doesn't work, it's frustrating – but the only thing to do is to find the bug(s) that are making your code not work. These bugs occur because the computer needs you to be as precise as possible while speaking any of its languages.

One method you can use to find bugs in your code is the rubber duck debugging technique.

So, what is the rubber duck debugging technique and how does it work? That’s what you will find out in this article.

# Why Do Programmers Use a Rubber Duck?

Many people who see a rubber duck on a programmer's desk might think it’s a toy or an inconsequential decoration.

In reality, the rubber duck is a tool that helps programmers solve problems. Programmers explain the quirks – often called bugs – in their code to the rubber duck which then helps them find a solution to the bug.

This is because explaining a problem to someone else helps you understand the problem better and find a solution to it. It allows you to see things from a new perspective and catch things you may have missed the first time around.

To understand the concept of rubber duck debugging better and the idea behind it, make sure you read this article to the end.

# What is Rubber Duck Debugging and How Does it Work?

The rubber duck debugging technique was popularized by a story from the 1999 book written by Andrew Hunt and David Thomas – The Pragmatic Programmer.

The story depicts how a programmer carries around a rubber duck and explains their codes to it line by line any time they encounter a bug.

Talking to an inanimate object about the bugs in your code might sound silly, but it’s very popular among programmers because it works.

In the rubber duck debugging technique, you don’t just explain the codes to the rubber duck line by line. You also tell the duck what exactly you’re trying to do and your goals for the code.

When you explain everything about the code to the rubber duck, you're much more likely to spot the bug hiding there.

Some of us have been in a situation where we encounter a problem, decide to explain the problem to a friend, and get the solution even when we are not done explaining the problem to that friend.

By explaining the bugs in your code to an inanimate object like a rubber duck, you don’t need to disturb anyone. You can also substitute a pet or other inanimate objects for the rubber duck.

# What is the Rubber Duck Theory?

The idea behind the rubber duck debugging technique is that when a programmer gets held back by bugs, the code is explained line by line to a rubber duck.

Think about this technique as a process in which you type a message to be posted on social media and read it out loud before you post it. This could help you spot the typographical and grammatical errors in it. The same thing is applicable to explaining your code to a rubber duck.

When you encounter bugs and you explain the code line by line to a rubber duck, the solution to the bugs often presents itself right from there.

# Final Thoughts

To solve problems and learn anything effectively, you need to talk about it and teach it. Rubber ducking helps you fix bugs because solutions are often found when you talk about a problem and teach it.

Over the years, rubber duck debugging has been a mainstay in problem-solving among software developers. According to some programmers, they even get rubber ducks as part of their welcome package when they get a new job.

It is not the rubber duck that tells you how to fix the bugs, of course. Rather, you figure out how you can fix the bugs by talking through what your code does line by line.

Rubber duck debugging, or talking to a rubber duck, is not exclusive to programming. You can also use this technique in other aspects of life.

## Source

By Kolade Kris, found at https://www.freecodecamp.org/news/rubber-duck-debugging/