**Day 13 - Debugging**

**Describe the problem**

It’s important to understand what’s going on and what the bug actually is only then can we start to tackle the issue

**Reproduce the bug**

Try to reproduce the bug. If you can reproduce the bug at will then you know exactly where the bug is and you’ll probably be able to work out what the issue is.

**Play computer and evaluate each line**

Go through the code line by line and evaluate what each line of code is supposed to be doing. A good idea is the pinpoint which block of code the bug is coming from first

**Compiler error and warning**

When coding, whichever IDE you are using will likely underline any potential mistakes while you are coding and will almost certainly show you any errors or warnings when trying to run the program. Listen to what the compiler is telling you and go and fix that bug. Sometimes the error codes can be confusing so might be a good idea to look it up if you are unsure.

**Use print!**

Whichever programming language you are using, there will be something similar to the print function. Use this function to output variables or use it as a marker to find out exactly where the code is breaking or why you aren’t getting the expected results.

**Use a debugger**

A debugger can be a very useful tool especially in larger projects. Why go through your code line by line in a big project on your own when you can go through it with your trusty debugger which will give you all the information you need along the way.

**Take a break**

Sometimes it’s good to take a break and step away from the screen. There have been several times that I have been banging my head against the wall wondering why some of my code isn’t working only to take a toilet break or go for a walk and instantly figure it out away from my screen.

**Ask a friend**

There’s never any shame in asking a friend, a colleague or a forum for help. Most of the time, people like to help out because it’s good practice for them, as developers, as well.

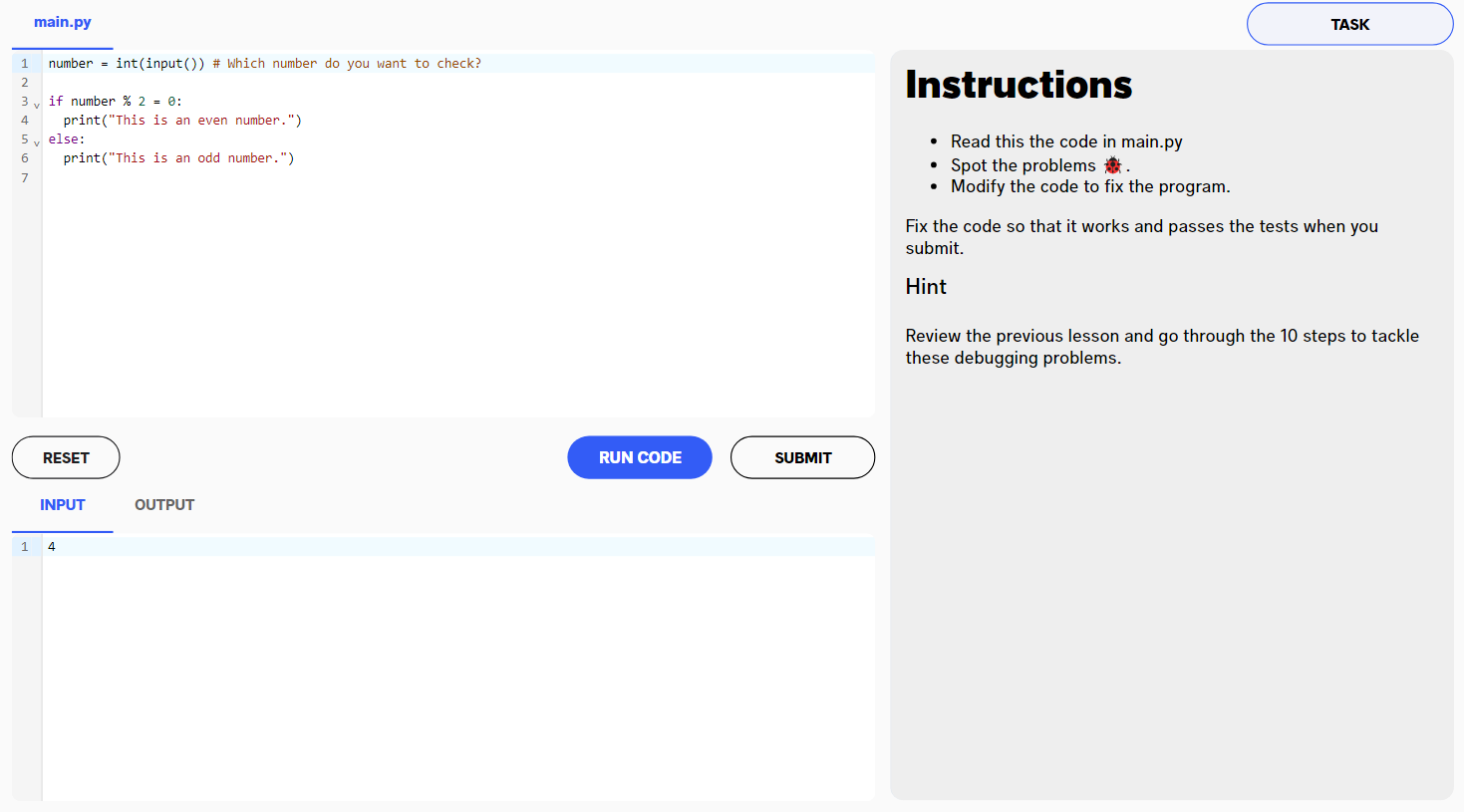
**Run often**

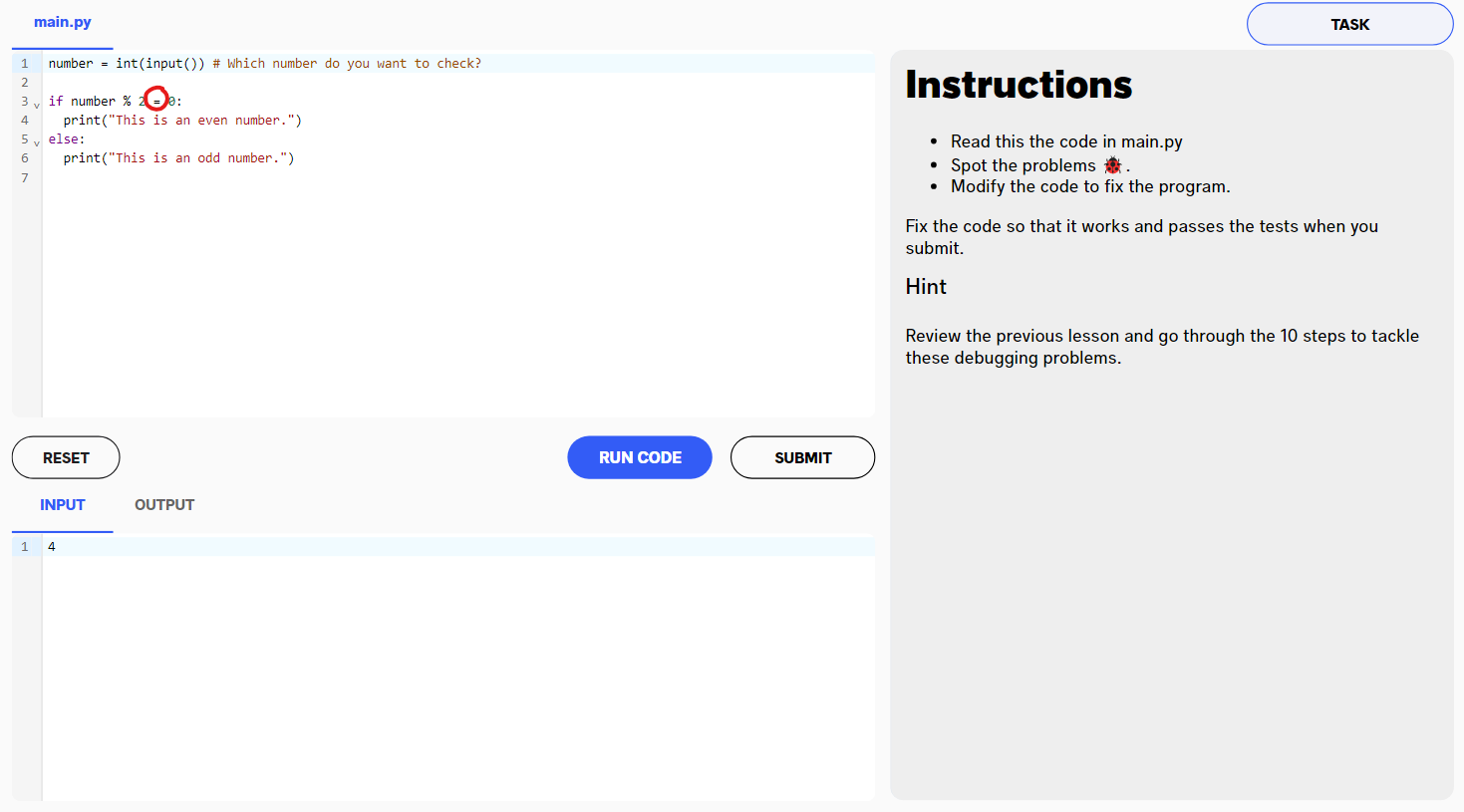
Run your program often. You don’t want to get to the end of a block of code you have been writing only to find out that it is littered with bugs and ambiguous outcomes. Running code often makes it easy to find bugs before they even become an issue.

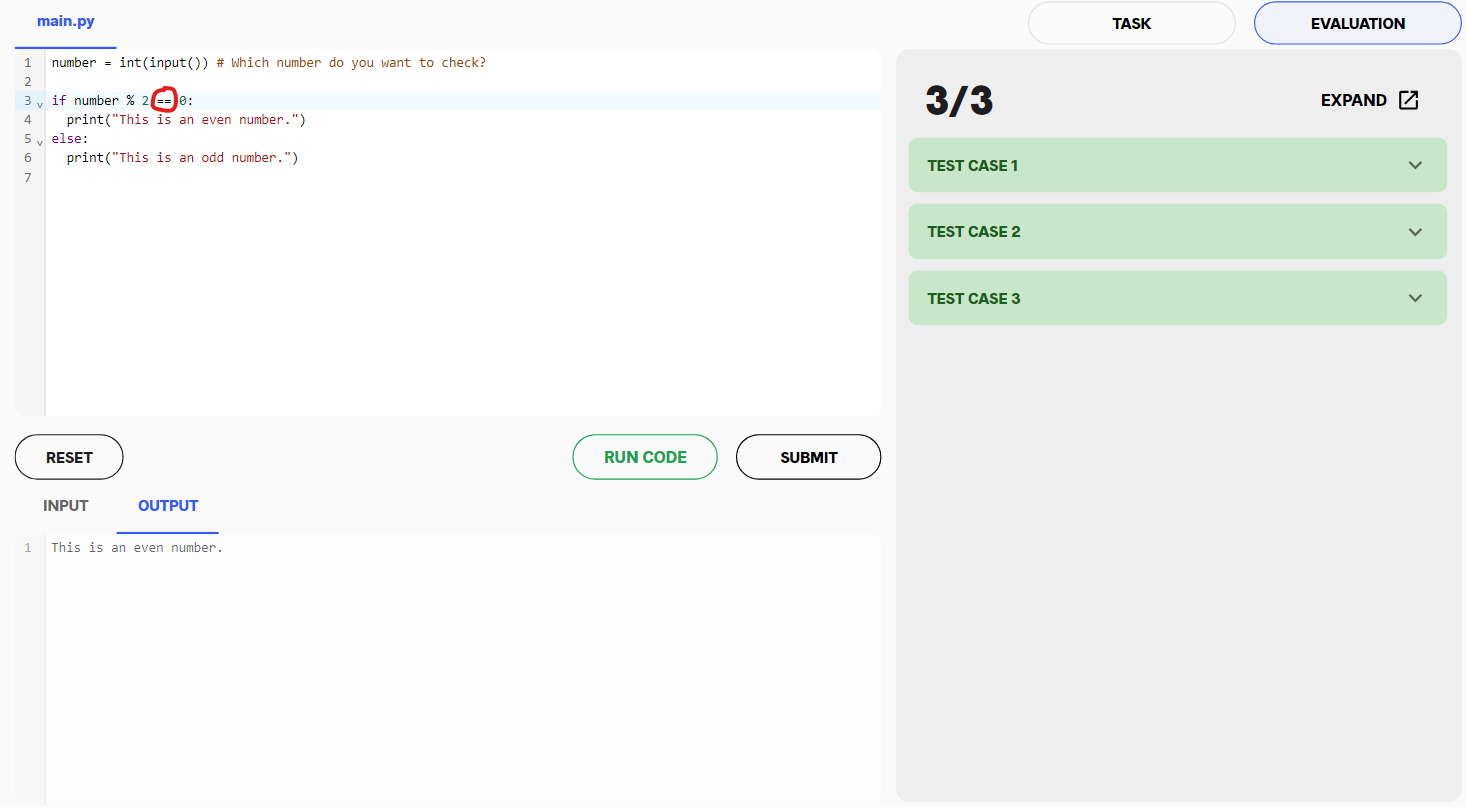
**Ask StackOverflow**

After all this and you’re still struggling, ask StackOverflow. However, be sure to make sure your question (or one similar) hasn’t already been asked, otherwise you might get some less than pleasant responses!

**Interactive Coding Exercise 1**

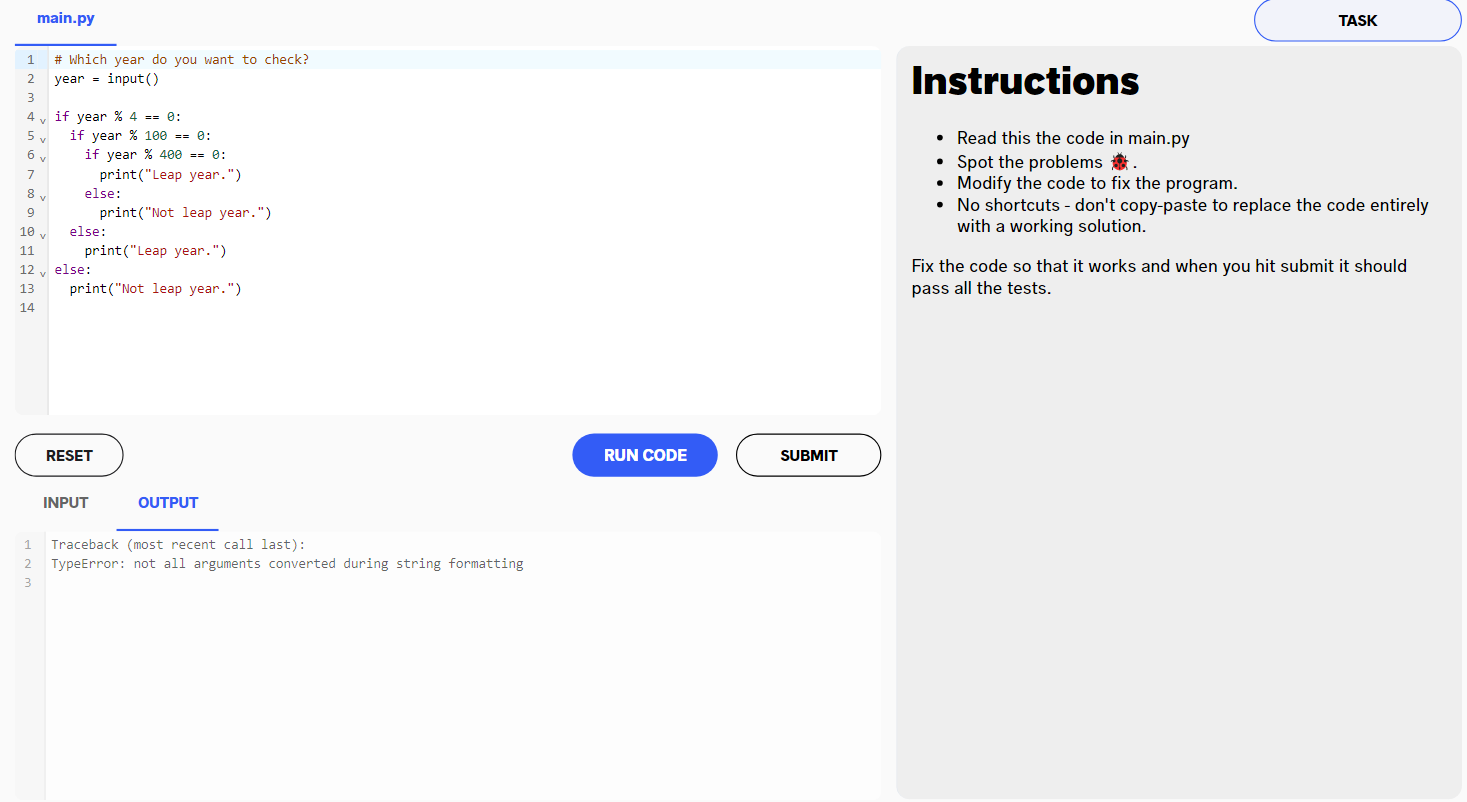


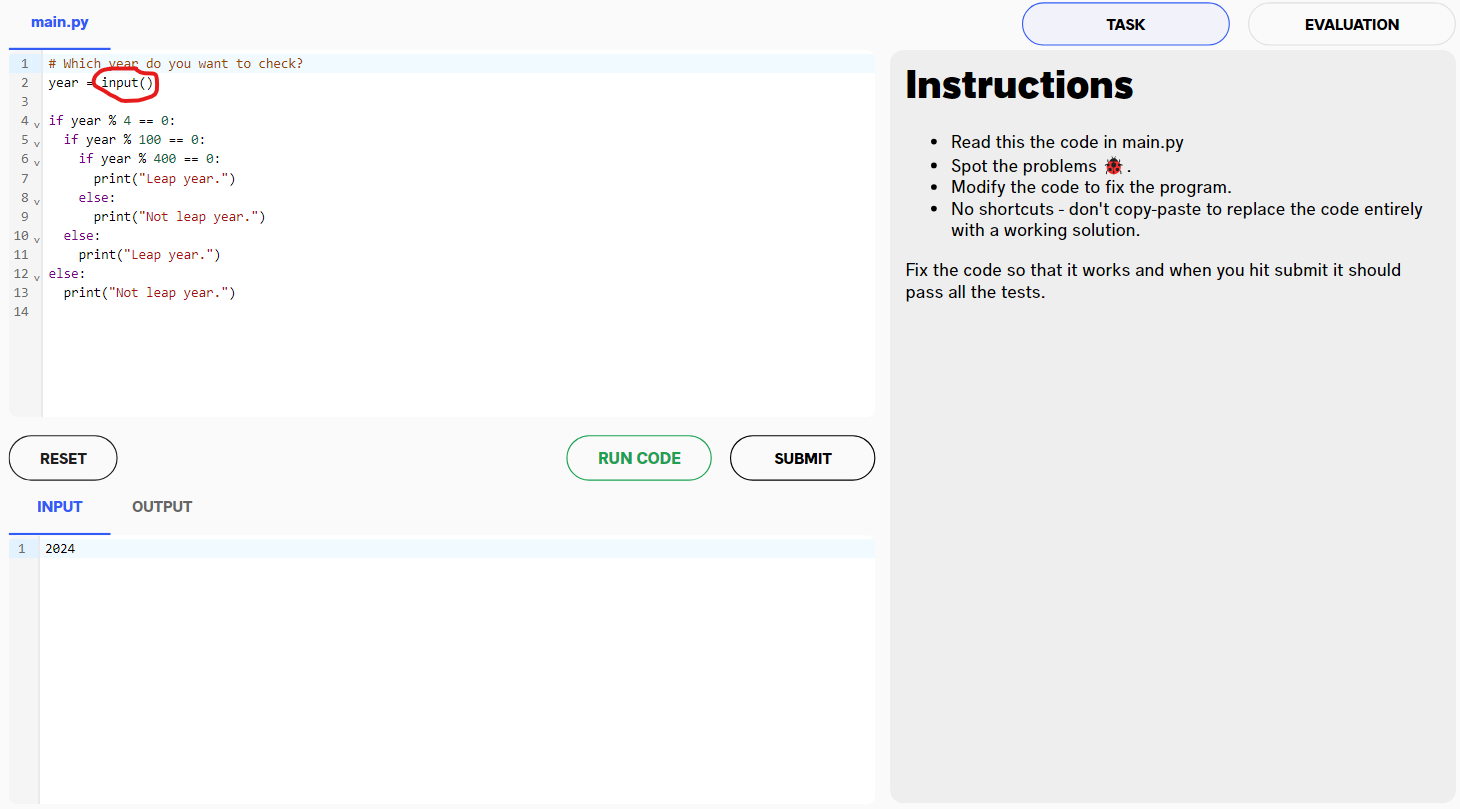


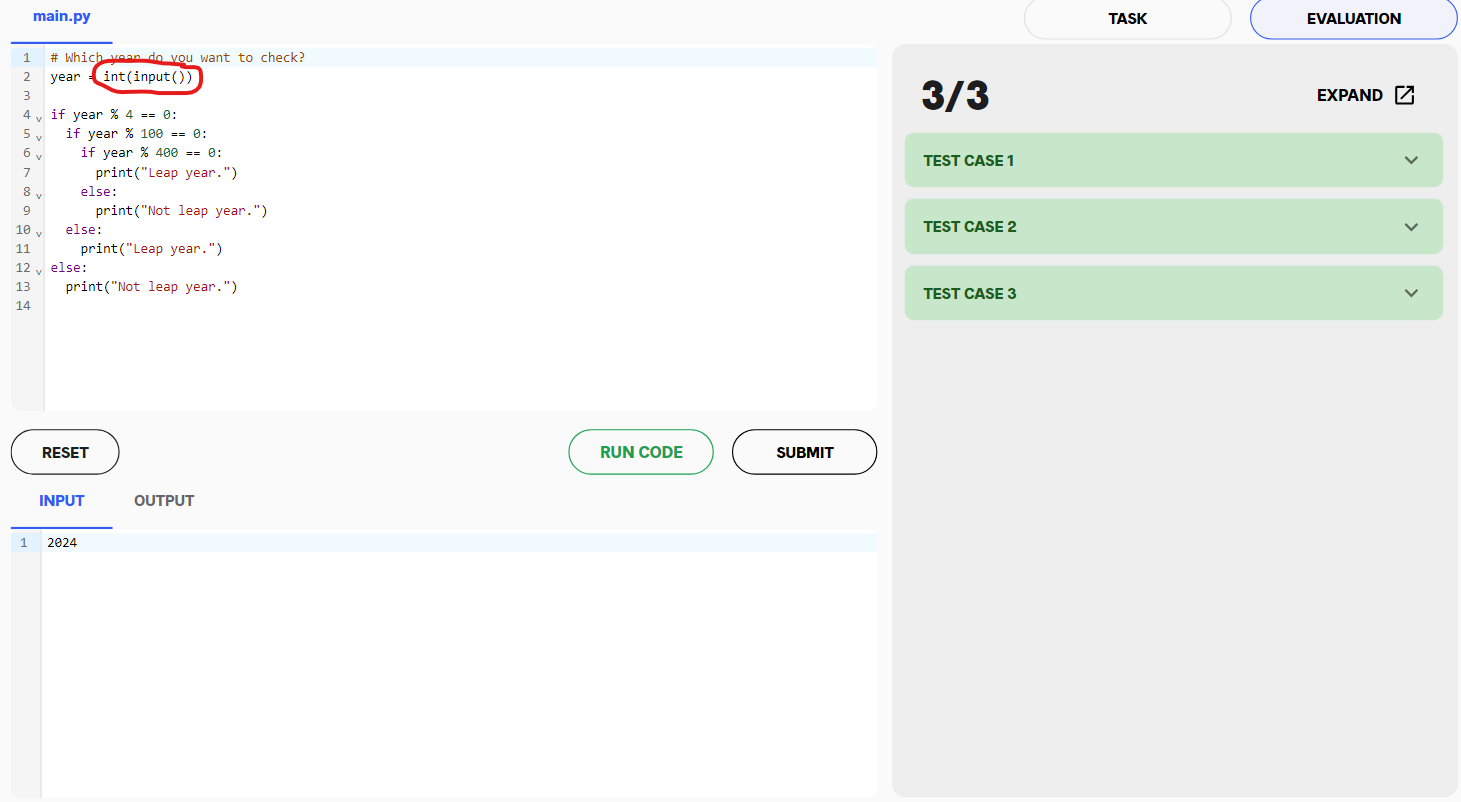


The bug in the above coding exercise was that rather than the equality operator being used, the assignment operator was used instead. This resulted in a syntax error. Switching to the equality operator fixed the bug and the program ran as intended

**Interactive Coding Exercise 2**

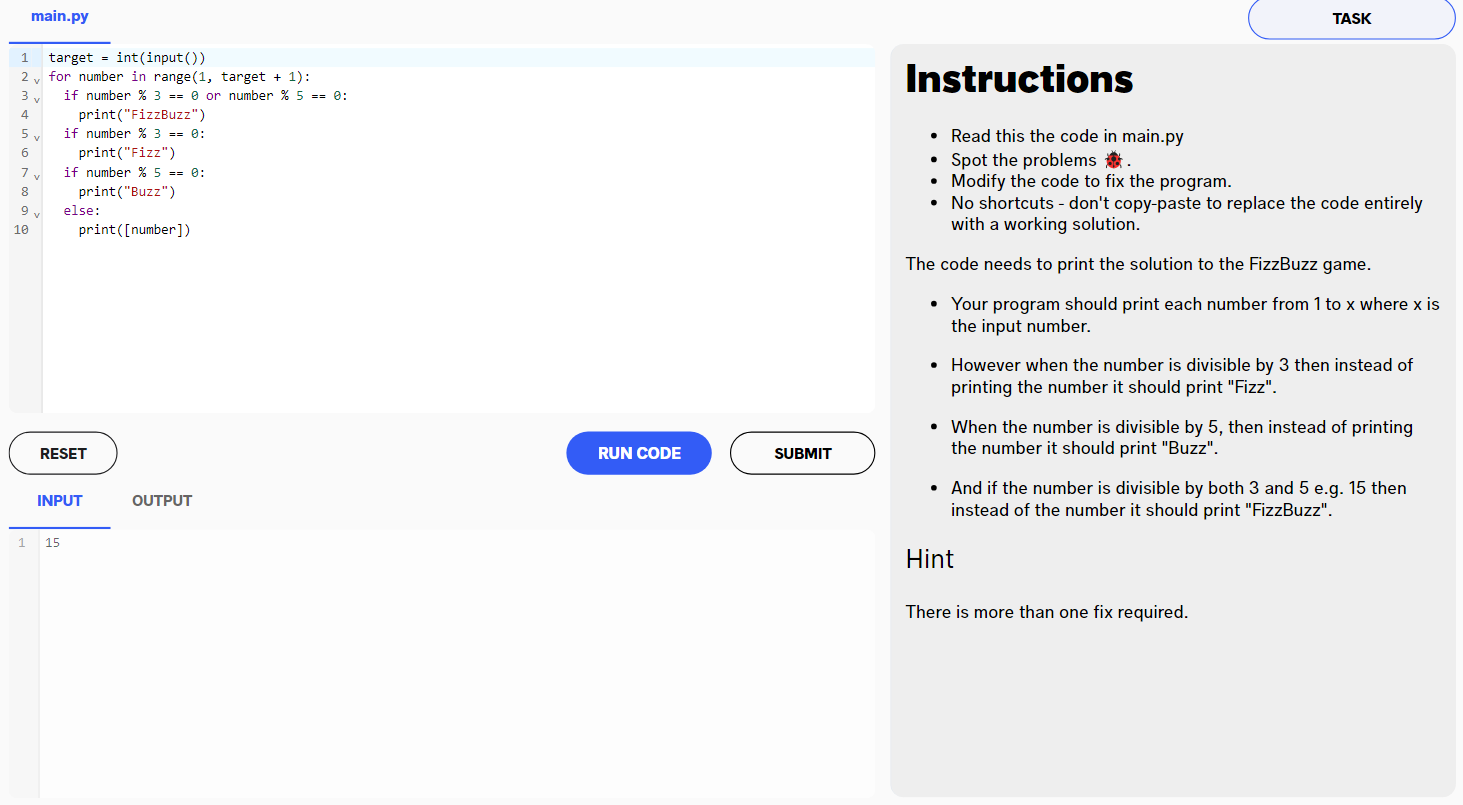


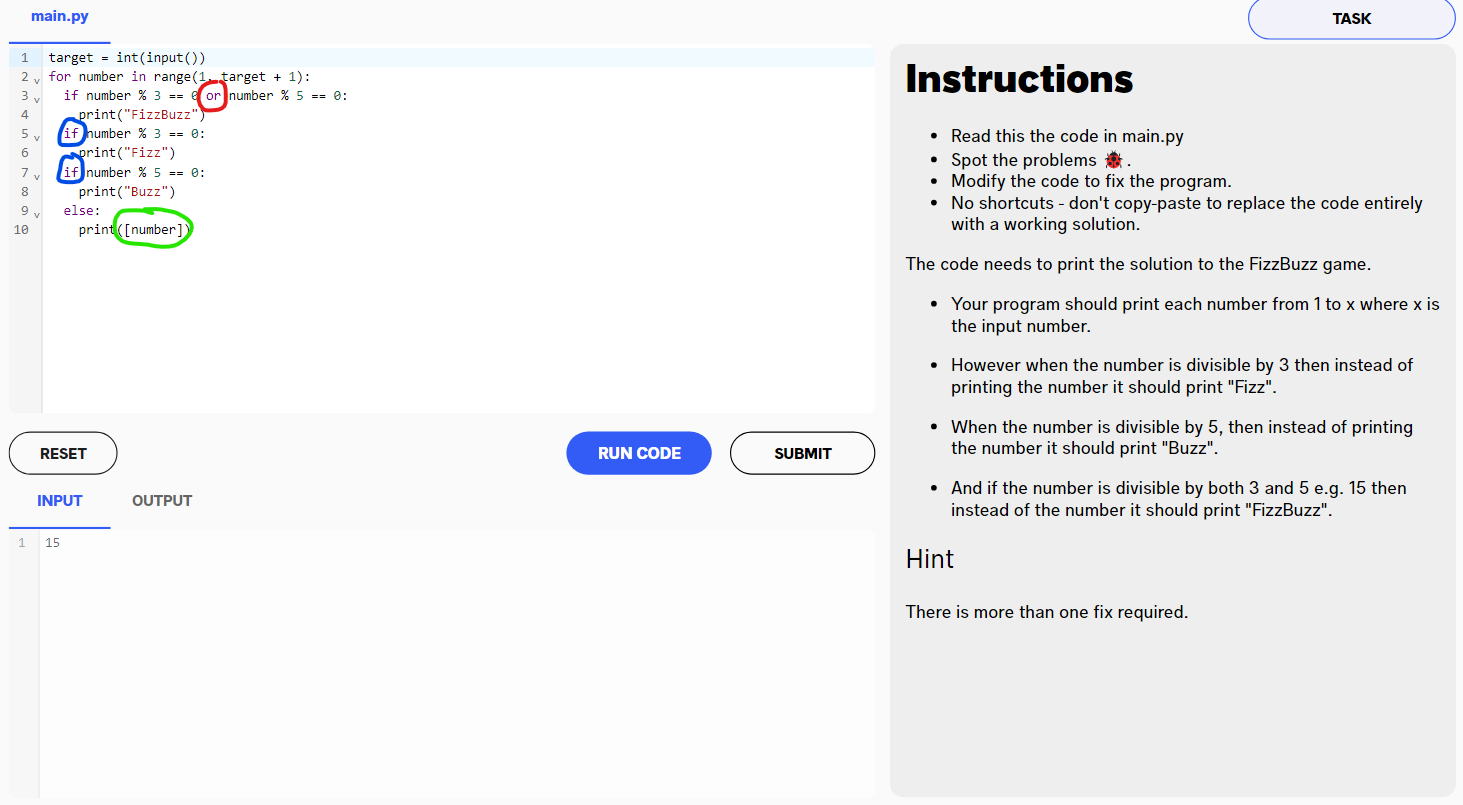


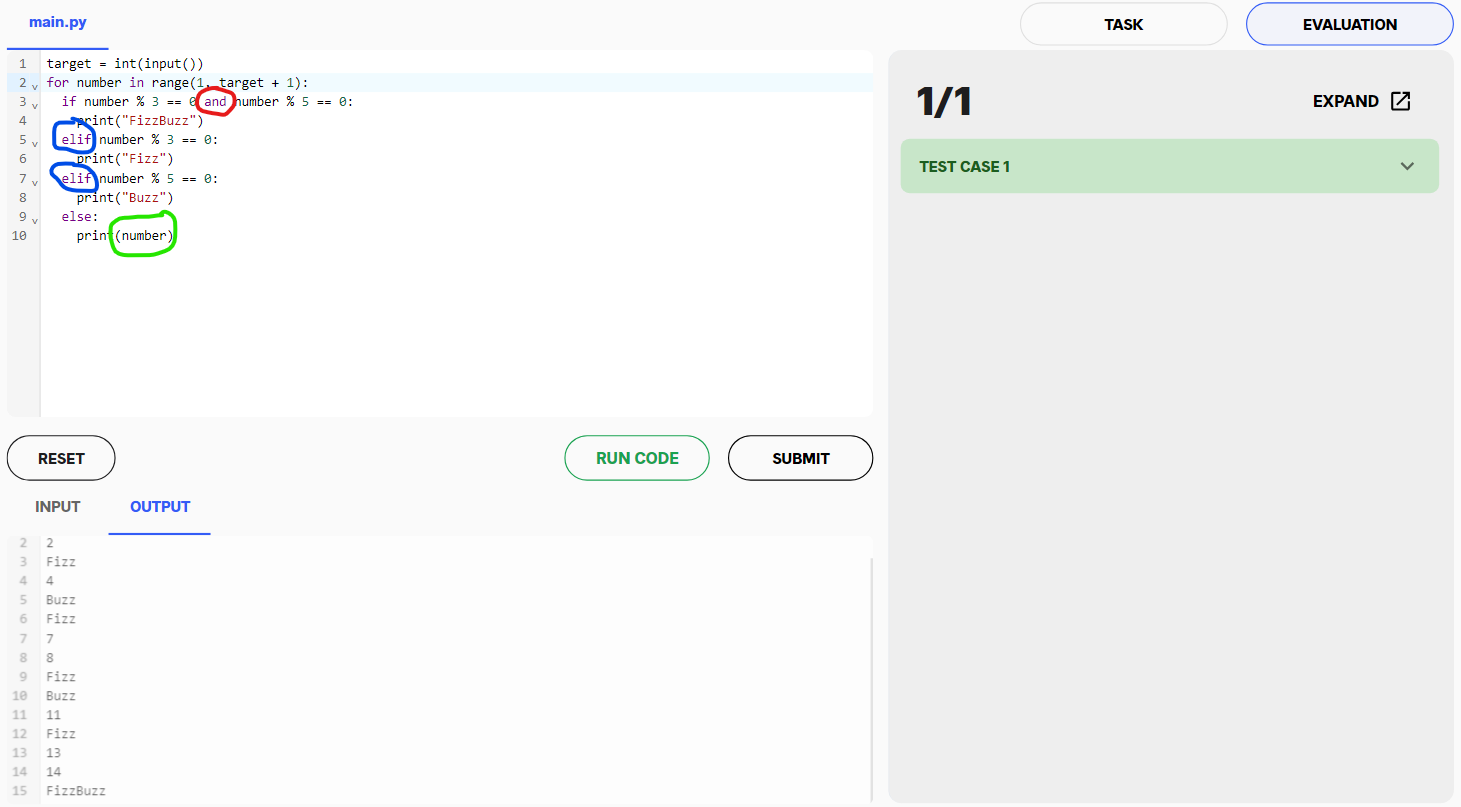


The above in the above coding exercise was that the input hadn't been type casted into an integer. From the original code, the input that was being stored inside ‘year’ was being stored as a string. The program later tried using mathematical operators on the string which resulted in a type error. Introducing int(input()) as seen in the code above ensures that ‘year’ is an int and thus mathematical operators can be used with it.

**Interactive Coding Exercise 3**

****

****

****

There were several bugs in the above coding exercise which I have colour coded.

Red - originally this was an or statement when in the program, we want this conditional statement to be true when BOTH conditions are met so this needs to be an ‘and’ statement, not a ‘for’ statement.

Blue - Both of these started off as if statements when they should have been else statements. This means that although both would still run, say we have the number 15 which is divisible by both 3 and 5. In this example, the output should be ‘FizzBuzz’. We do get ‘FizzBuzz’ but that isn’t the only output we get. Firstly, line 3 would run and return true thus printing ‘FizzBuzz’. After this, line 5 would run and would return true and print ‘Fizz’. Lastly, line 7 would execute and return true and so would print ‘Buzz’. Instead of just one output, we end up with all three. This is because each if statement is being executed when in reality, we only want these statements executing when the previous statement has returned false and to achieve this, we need to use ‘elif’ rather than starting another if statement.

Green - Green is originally returning a list with the current iteration of number in it. This isn’t a huge issue as the output is still intuitive, however, it isn’t necessary and the iteration of the ‘number’ variable can just be printed on its own.