Create — Applications from Ideas
Written Response Submission Template

Program Purpose and Development

2a)

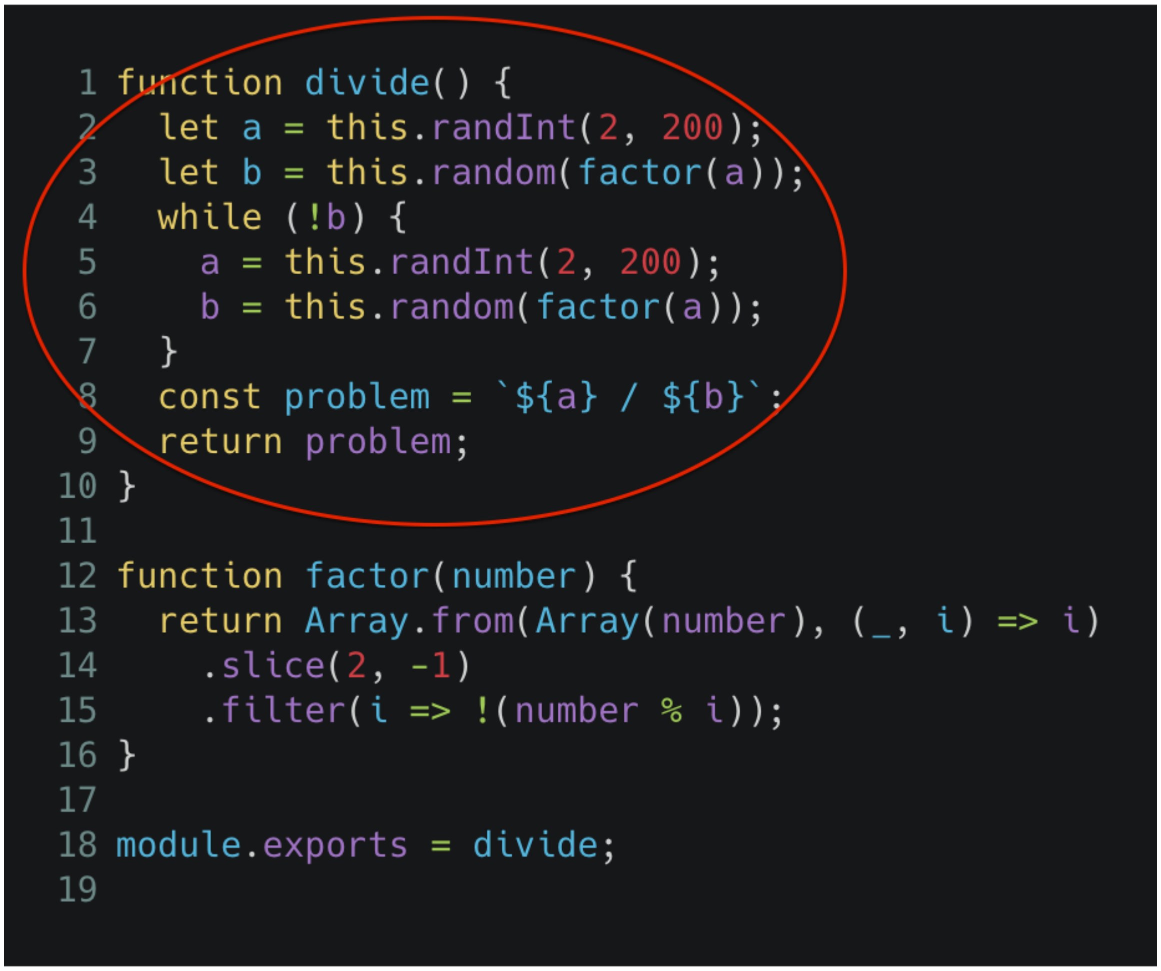
My program is a command line-based game randomly generating simple mathematical expressions for players to solve. Written in Node.js, a JavaScript runtime based on Chrome's V8 engine, my program creates problems involving addition, subtraction, multiplication, and division. All questions only involve two small positive integers in order to make all problems quickly solvable while maintaining their randomized qualities. In order to win the game, players must enter the correct answer to the generated problem into their command line interface within fifteen seconds of the program's initiation. If they do not, my program tells players that they have lost, reveals the correct answer, and then terminates.

2b)

While I developed my program, I found it difficult to create an interactive game with the Node.js standard input stream and wanted a higher-level interface for working with user input to speed up project development. Thus, I searched for an external command prompt-handling module to import. I first experimented with the **inquirer** module, but its prompts could only accept user input once and could not reject incorrect inputs to ask users to retry inputting a correct response. Later, I discovered that the **prompt** module satisfied my needs because it offered unique input validator functionality.

Another difficulty my program faced was determining the answer to the expression it had manually generated. In my program, an algorithm randomly generates two integers separately and combines them with a random mathematical operation to be displayed as a string expression. My original function for evaluating answers was a switch statement with statements for each type of mathematical operation using the only parameters available: the first integer, second integer, and the type of operation the expression used. However, after reading the ECMAScript specification more thoroughly, I realized I could the **eval()** function to evaluate the mathematical expression as a string. By refactoring my code to use the **eval()** function, I eliminated a need to make a complex function for evaluating my random math expressions.

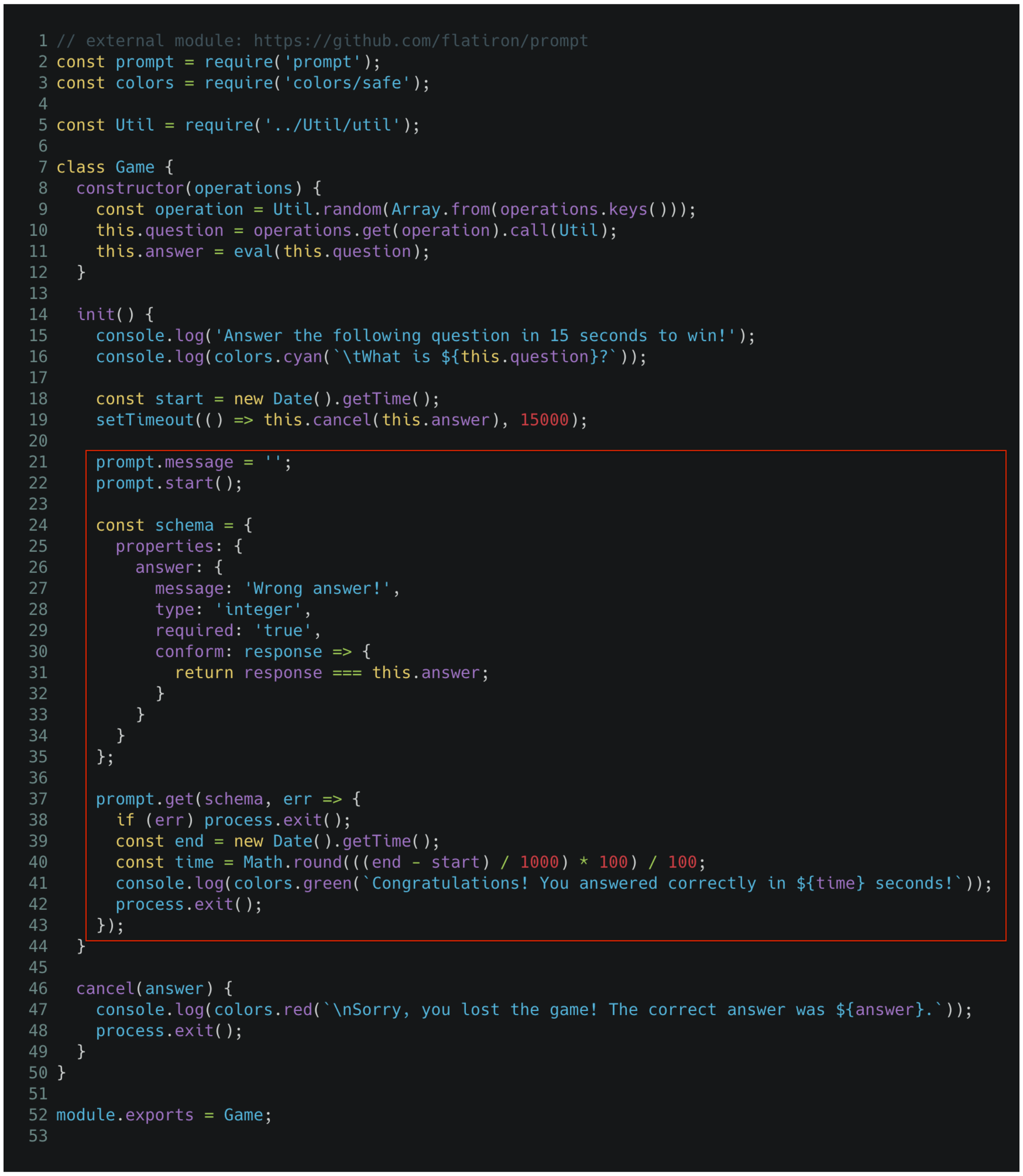
2c)



```
function divide() {
  let a = this.randInt(2, 200);
  let b = this.random(factor(a));
  while (!b) {
    a = this.randInt(2, 200);
    b = this.random(factor(a));
  }
  const problem = `${a} / ${b}`;
  return problem;
}

function factor(number) {
  return Array.from(Array(number), (_, i) => i)
    .slice(2, -1)
    .filter(i => !(number % i));
}

module.exports = divide;

```

The **divide()** algorithm returns a string representing a random division expression after my program determines the random operation it will use. It is called so that the this keyword is defined as an utility class with the **randInt()** and **random()** methods. First, variable **a** is defined as a random number between 2 and 200 generated by **randInt()**. Next, **a** is passed as the **number** parameter to the **factor()** function to find all of its positive integer factors. To do this, I create an array with the length of **number** and set the value of all its elements to its index in the array and exclude the first two and last elements from the array because I don't want to include 0, 1, or the original number as factors. Then, **factor()** eliminates numbers that aren't factors of **number** using the **modulus (%)** mathematical operator, which gives the remainder between two numbers. Since **0** is the only numerical falsey value, I use the **! (NOT)** logical operator to filter out all dividends of **number** that do not give a remainder of 0. One element from the generated array is randomly selected by **random()** and set as the value of variable **b**. In the event that **a** is a prime number, **b** will have a null value, so I use a **while** loop to regenerate new values for **a** and its factor **b** until **b** is a valid integer. Finally, I create a string representing a new division expression using the values of **a** and **b**, and the **divide()** algorithm returns that value to the algorithm that originally called it.

2d)



```javascript
// external module: https://github.com/flatiron/prompt
const prompt = require('prompt');
const colors = require('colors/safe');

const Util = require('../Util/util');

class Game {
  constructor(operations) {
    const operation = Util.random(Array.from(operations.keys()));
    this.question = operations.get(operation).call(Util);
    this.answer = eval(this.question);
  }

  init() {
    console.log('Answer the following question in 15 seconds to win!');
    console.log(colors.cyan(`\tWhat is ${this.question}?`));

    const start = new Date().getTime();
    setTimeout(() => this.cancel(this.answer), 15000);

    prompt.message = '';
    prompt.start();

    const schema = {
      properties: {
        answer: {
          message: 'Wrong answer!',
          type: 'integer',
          required: 'true',
          conform: response => {
            return response === this.answer;
          }
        }
      }
    };

    prompt.get(schema, err => {
      if (err) process.exit();
      const end = new Date().getTime();
      const time = Math.round(((end - start) / 1000) * 100) / 100;
      console.log(colors.green(`Congratulations! You answered correctly in ${time} seconds!`));
      process.exit();
    });
  }

  cancel(answer) {
    console.log(colors.red(`\nSorry, you lost the game! The correct answer was ${answer}.`));
    process.exit();
  }
}

module.exports = Game;
```

My abstraction is the event listener **prompt.get(),** created using the external library **prompt** to represent a stream of valid responses to my prompt. In the **schema** variable, I define several properties for a valid input. Through the **conform** property, I use the **strict equality (===) logical comparison operator** in a function to only validate inputs that are equivalent to the generated answer.

After the prompt is initiated, the **schema** variable is then passed to the **prompt.get()** event listener as a parameter; it fires once valid input is received calculate so I can calculate how many seconds it took between the prompt initiation and the successful input submission. I do this by converting milliseconds to seconds while rounding off at two decimal places using **Math.round()** and several mathematical operations. Through this abstraction, I simplify the complexity of my program by creating a higher-level interface for interacting with Node.js standard input streams that only accepts valid inputs according to a strict specification.