Aim:- WAP to implement concept of REPL in commandline

LO Mapped:- LO6

Theory:-

The **node:repl** module exports the repl.REPLServer class. While running, instances of repl.REPLServer will accept individual lines of user input, evaluate those according to a user-defined evaluation function, then output the result. Input and output may be from stdin and stdout, respectively, or may be connected to any Node.js stream.

Instances of **repl.REPLServer** support automatic completion of inputs, completion preview, simplistic Emacs-style line editing, multi-line inputs, ZSH-like reverse-i-search, ZSH-like substring-based history search, ANSI-styled output, saving and restoring current REPL session state, error recovery, and customizable evaluation functions. Terminals that do not support ANSI styles and Emacs-style line editing automatically fall back to a limited feature set.

Commands and special keys

The following special commands are supported by all REPL instances:

.break: When in the process of inputting a multi-line expression, enter the .break command (or press Ctrl+C) to abort further input or processing of that expression.

.clear: Resets the REPL context to an empty object and clears any multi-line expression being input.

.exit: Close the I/O stream, causing the REPL to exit.

.help: Show this list of special commands.

.save: Save the current REPL session to a file: > .save ./file/to/save.js

.load: Load a file into the current REPL session. > .load ./file/to/load.js

.editor: Enter editor mode (Ctrl+D to finish, Ctrl+C to cancel).

By default, all instances of repl.REPLServer use an evaluation function that evaluates JavaScript expressions and provides access to Node.js built-in modules. This default behavior can be overridden by passing in an alternative evaluation function when the repl.REPLServer instance is created.

Code:-

Calculator.js

```
const readline = require('readline');

const rl = readline.createInterface({
input: process.stdin,
output: process.stdout,
});

function add(x, y) {
return x + y;
}

function subtract(x, y) {
return x - y;
}
```

```
function multiply(x, y) {
return x * y;
}

function divide(x, y) {
if (y === 0) {
return "Cannot divide by zero";
}
return x / y;
}

function startCalculator() {
console.log("Simple Calculator");
rl.question("Enter 'exit' to end the program or press Enter to continue: ",
(answer) => {
if (answer === 'exit') {
rl.close();
return;
}

rl.question("Enter operation (add/subtract/multiply/divide): ", (operation) => {
if (['add', 'subtract', 'multiply', 'divide'].includes(operation)) {
rl.question("Enter first number: ", (num1) => {
rl.question("Enter second number: ", (num2) => {
num1 = parseFloat(num1);
num2 = parseFloat(num2);

switch (operation) {
case 'add':
console.log(`Result: ${add(num1, num2)}`);
break;
case 'subtract':
console.log(`Result: ${subtract(num1, num2)}`);
break;
case 'multiply':
console.log(`Result: ${multiply(num1, num2)}`);
break;
case 'divide':
console.log(`Result: ${divide(num1, num2)}`);
break;
default:
console.log("Invalid operation");
}
```

```
startCalculator();
});
});
} else {
console.log("Invalid operation");
startCalculator();
}
});
});
}

startCalculator();
```

Output:-

```
karani@Karanis-MacBook-Air src % node calculator.js
Simple Calculator
Enter 'exit' to end the program or press Enter to continue:
Enter operation (add/subtract/multiply/divide): add
Enter first number: 4
Enter second number: 6
Result: 10
Simple Calculator
Enter 'exit' to end the program or press Enter to continue:
Enter operation (add/subtract/multiply/divide): subtract
Enter first number: 3
Enter second number: 6
Result: -3
Simple Calculator
Enter 'exit' to end the program or press Enter to continue: exit
karani@Karanis-MacBook-Air src %
```

Conclusion:-

Learnt about REPL in node.js , explored more about its command and special keys and created a calculator in commandline using REPL