Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 1**	
Title of Lab Assignment: Understand the concepts of REPL and Node.js console.			
DOP: 12-09-2023		**DOS: 12-09-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO3, PO5, PSO1, PSO2**	**Signature:**	**Marks:**

Practical 1

Aim:

1) To understand the fundamentals of Node.js, its advantages, disadvantages, and key concepts like the Node.js process model and traditional web server model Installation.
2) Print Today's date and time using REPL.
3) Write a program to Print the given pattern:

   1 2 3 4 5
   1 2 3 4
   1 2 3
   1 2
   1
4) Write a Program to print the first 20 Fibonacci numbers (Take input through the command line).
5) Write a program to print prime nos between 1 to 100.

Theory:

1) **Introduction to Node.js**

   Node.js, an open-source and cross-platform JavaScript runtime environment, is constructed upon Google Chrome's V8 JavaScript engine. Its primary purpose is to facilitate the execution of JavaScript code beyond the confines of web browsers, rendering it a robust choice for server-side development. Node.js boasts an event-driven, non-blocking architecture, which endows it with the capability to efficiently manage numerous concurrent connections. These attributes make it a formidable player in modern web development.

2) **Advantages of Node.js:**

   Highly Efficient: Node.js employs an event-driven, non-blocking I/O model, a hallmark of its efficiency in managing concurrent requests. This model ensures that Node.js can adeptly handle multiple tasks simultaneously without performance bottlenecks.

Single Language: One of Node.js' remarkable strengths is that it allows developers to utilize JavaScript for both client-side and server-side scripting. This unification of programming languages streamlines the development process and eliminates the need to switch between languages.

Vast Ecosystem: Node.js boasts an extensive ecosystem of libraries and packages available through npm (Node Package Manager), simplifying development by providing readily accessible resources.

Scalability: Designed with scalability in mind, Node.js excels at accommodating a substantial number of concurrent connections, making it an excellent choice for applications that expect high traffic.

Real-time Applications: Node.js is the ideal platform for crafting real-time applications like chat applications, online gaming platforms, and live data streaming services, thanks to its event-driven, non-blocking nature.

3) Disadvantages of Node.js:

Single-threaded: Node.js operates on a single-threaded event loop, which can lead to performance issues for CPU-bound tasks, limiting its suitability for certain types of applications.

Callback Hell: Managing asynchronous code using callbacks can result in a phenomenon known as "callback hell," where code becomes complex and challenging to read, potentially hampering development efficiency.

Lack of In-built Modules: Some modules available in other programming languages may not have readily available counterparts in the Node.js ecosystem, necessitating additional development efforts.

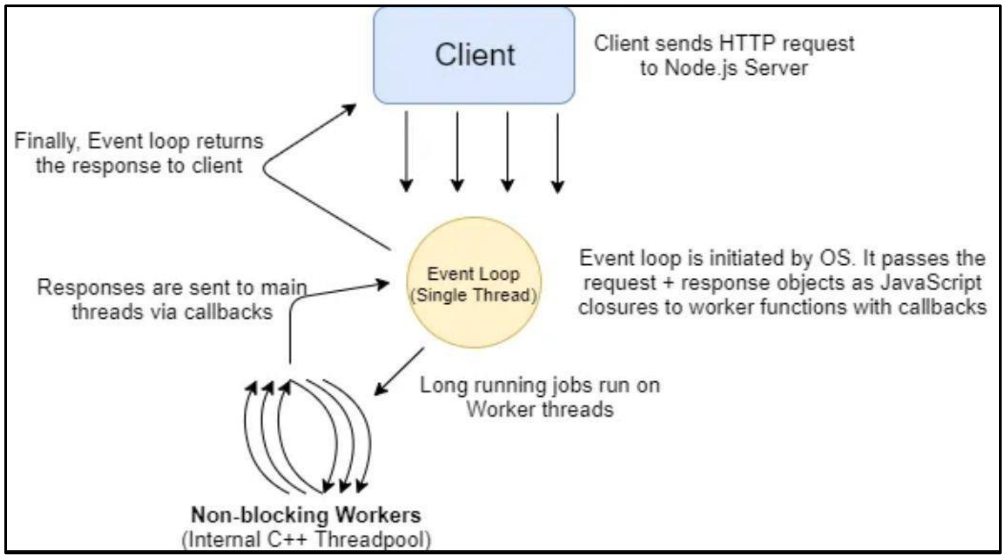
Less Suitable for Large-scale Applications: While Node.js is highly capable, it may not be the best choice for very large and intricate applications that require complex orchestration and multi-threaded processing.

4) Node.js Process Model:

Node.js adopts a unique process model that distinguishes it from traditional web servers. In Node.js, all code runs within a single process, and requests are handled on a single thread. This approach offers several advantages, including reduced resource consumption and improved scalability.
When a request arrives in a Node.js application, it doesn't block the main thread. Instead, it's placed in an event queue. Node.js employs an event loop, a critical component, to continuously monitor this queue for events, such as incoming requests or completed asynchronous operations. This event loop efficiently manages the flow of requests and ensures that the server remains responsive.

Non-Blocking Requests: These are requests that don't involve time-consuming computations or data retrievals. In such cases, Node.js can immediately process the request, construct the response, and send it back to the client without waiting.

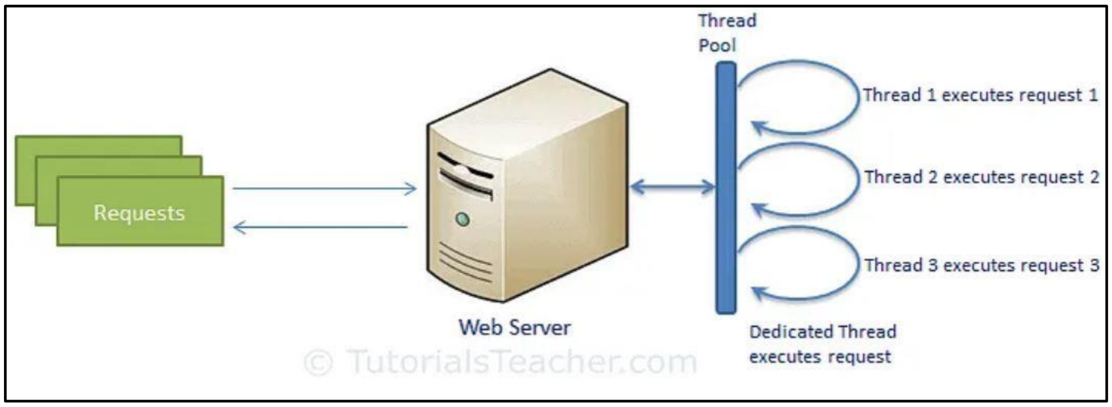
Blocking Requests: Some requests require I/O operations, like reading from a file or querying a database. These operations can take a significant amount of time. In Node.js, blocking requests are offloaded to a worker thread pool. Each request sent to the pool is associated with a callback function, which is executed when the operation completes. This means that while one thread is waiting for I/O to finish, the main thread can continue processing other requests, making the most of the available resources.


Client
Client sends HTTP request to Node.js Server
Finally, Event loop returns the response to client
Event Loop (Single Thread)
Responses are sent to main threads via callbacks
Event loop is initiated by OS. It passes the request + response objects as JavaScript closures to worker functions with callbacks
Long running jobs run on Worker threads
Non-blocking Workers (Internal C++ Threadpool)

5) Web Server Model:

Imagine a traditional web server as a team of workers. Each worker (or thread) can handle one task at a time. When a new task (or request) arrives, it gets assigned to an available worker. However, if all the workers are currently busy with other tasks, the new task must wait its turn. It's like waiting in line at a busy store; you can't be helped until the cashier is available.

This way of handling requests is called synchronous or blocking because tasks are processed in a sequential, step-by-step fashion. It's like a single lane road where cars must wait for the one in front to move before they can proceed. This approach ensures that requests are processed in order but can sometimes result in delays if there are many tasks or if some tasks take a long time to complete.


Requests
Web Server
Thread Pool
Thread 1 executes request 1
Thread 2 executes request 2
Thread 3 executes request 3
Dedicated Thread executes request
© TutorialsTeacher.com

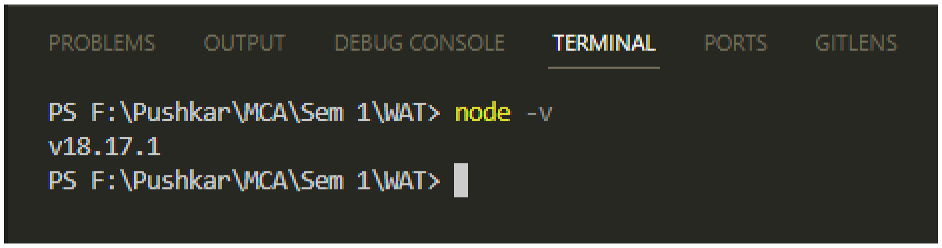
6) Installation:

Step 1: Downloading the Node.js '.msi' installer.

Step 2: Running the Node.js installer.

Step 3: Check Node.js and NPM Version

Command: node -v



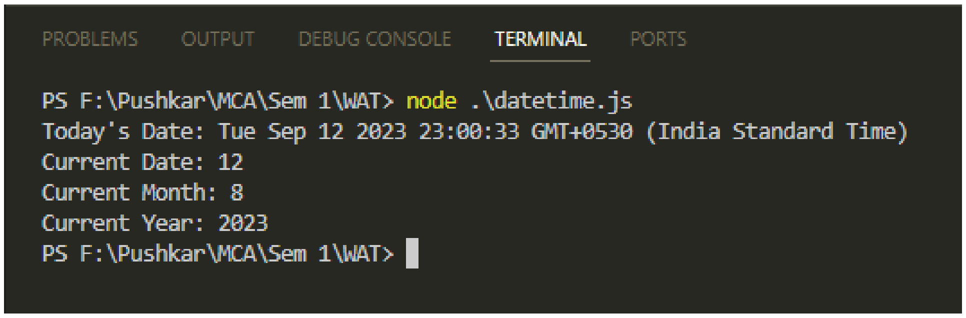
```
PS F:\Pushkar\MCA\Sem 1\WAT> node -v
v18.17.1
PS F:\Pushkar\MCA\Sem 1\WAT>
```

1. **Print today's date and time using REPL.**

Code:

```
const today = new Date();
console.log("Today's Date: " + today);
console.log("Current Date: " + today.getDate());
console.log("Current Month: " + today.getMonth());
console.log("Current Year: " + today.getFullYear());
```

Output:



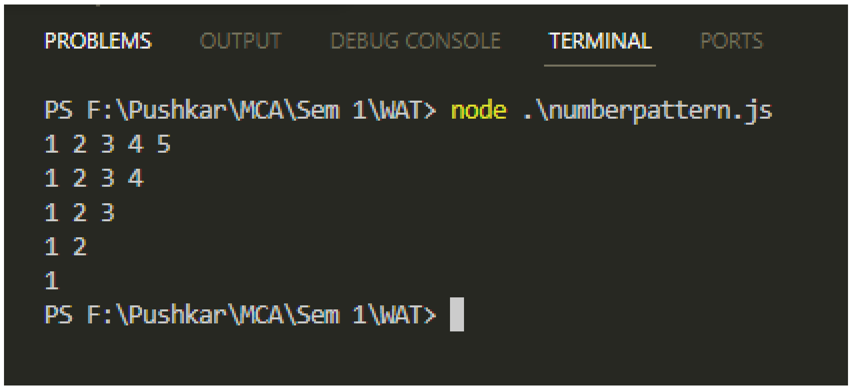
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\datetime.js
Today's Date: Tue Sep 12 2023 23:00:33 GMT+0530 (India Standard Time)
Current Date: 12
Current Month: 8
Current Year: 2023
PS F:\Pushkar\MCA\Sem 1\WAT>
```

2. Write a program to print a given pattern.

Code:

```
function printPattern(rows) {
   for (let i = rows; i >= 1; i--) {
      let pattern = '';
      for (let j = 1; j <= i; j++) {
         pattern += j + ' ';
      }
      console.log(pattern);
   }
}
const numRows = 5;
printPattern(numRows);
```

Output:



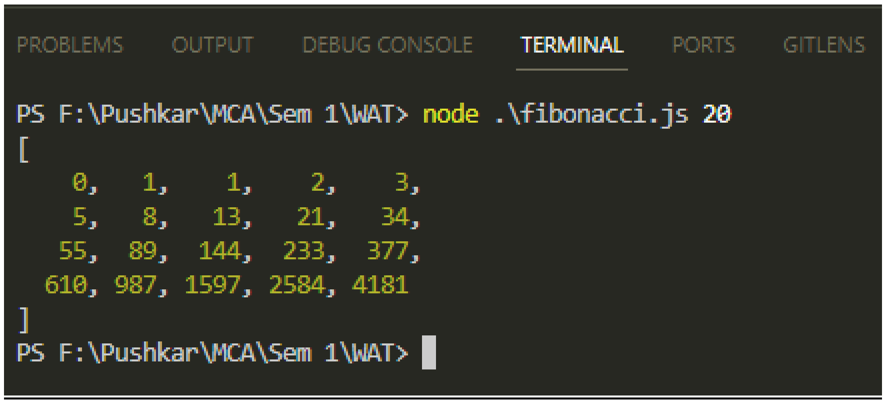
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\numberpattern.js
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
PS F:\Pushkar\MCA\Sem 1\WAT>
```

3. **Write a program to print the first 20 fibonacci numbers (take input through command line).**

Code:

```
const n = parseInt(process.argv[2]);
function fibonacci(n){
   let fib = [0, 1];
   for(let i = 2; i < n; i++){
      fib[i] = fib[i - 1] + fib [i - 2];
   }
   return fib;
}
console.log(fibonacci(n));
```

Output:



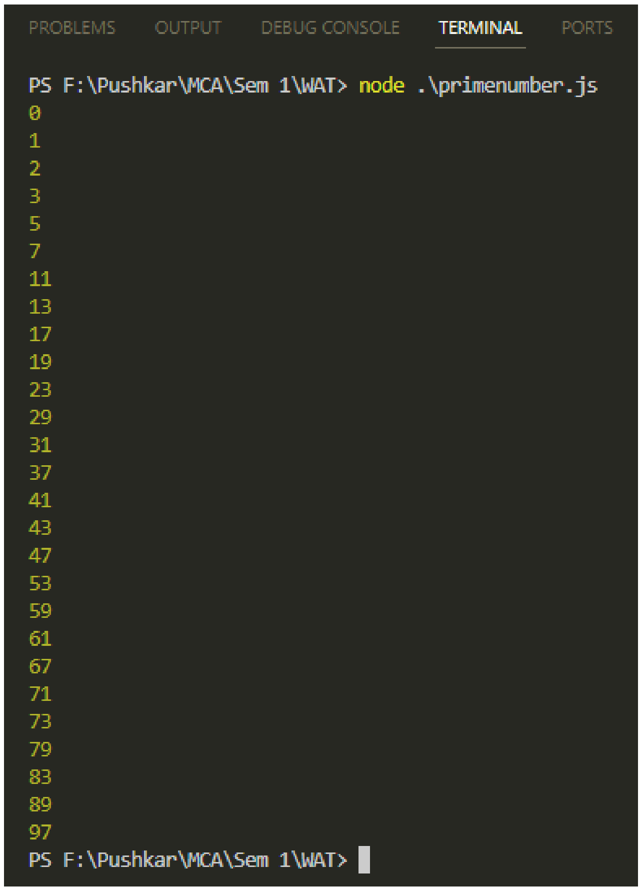
```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\fibonacci.js 20
[
    0,   1,    1,    2,    3,
    5,   8,   13,   21,   34,
   55,  89,  144,  233,  377,
  610, 987, 1597, 2584, 4181
]
PS F:\Pushkar\MCA\Sem 1\WAT>
```

4. Write a program to print numbers between 0 and 100.

Code:

```
for (var num = 0; num <= 100; num++) {
   var notPrime = false;
   for (var i = 2; i <= num; i++) {
      if (num % i == 0 && i != num) {
         notPrime = true;
      }
   }
   if (notPrime == false) {
      console.log(num);
   }
}
```

Output:



```
PS F:\Pushkar\MCA\Sem 1\WAT> node .\primenumber.js
0
1
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
PS F:\Pushkar\MCA\Sem 1\WAT>
```

Conclusion:

Implemented various concepts of REPL and performed various Node.js programs.