The numbers 1, 4, 3, and 2 will be logged to the console when the code is executed.

Let's break down the execution step-by-step:

1. `console.log(1);`: This line logs the number 1 to the console immediately.

2. `setTimeout(function(){console.log(2)}, 1000);`: This sets up a timer to log the number 2 to the console after a delay of 1000 milliseconds (1 second). However, it doesn't block the code execution, so the program continues to the next statement.

3. `setTimeout(function(){console.log(3)}, 0);`: This sets up a timer to log the number 3 to the console after a delay of 0 milliseconds. Even though the delay is 0, it will still be executed asynchronously after other synchronous tasks are completed.

4. `console.log(4);`: This line logs the number 4 to the console immediately after the number 1.

Now, let's analyze the output order:

- `1` is logged immediately when the first `console.log(1);` statement is executed.

- `4` is logged immediately after the number 1 because the code is synchronous, and it executes sequentially.

- `3` is logged next, even though it has a delay of 0 milliseconds. The reason is that the `setTimeout` function with a delay of 0 still gets pushed to the end of the event loop queue. Since the current code execution is finished, it goes to the next task in the queue, which is the `setTimeout` with 0 delay, and logs the number 3.

- `2` is logged last. It has a delay of 1000 milliseconds, so it will be executed approximately 1 second after the current code finishes executing. During this delay, the number 2 is pushed to the event loop queue and then logged after the previous tasks are completed.

Therefore, the final order of logs will be:

```

1

4

3

2

```