(Notes) Event Loop

Question 1 - What is Event Loop?
Explanation:
The Event Loop in JavaScript is a mechanism responsible for managing asynchronous behavior in a single-threaded environment. It acts like a traffic controller, ensuring tasks are executed in an orderly manner by processing pending tasks in queues (microtasks and macrotasks).

Question 2 - Why do we need the event loop to manage these task queue and microtask queue?
Explanation:
The event loop is necessary to handle asynchronous operations in JavaScript effectively. It manages task queues and microtask queues to ensure that tasks are executed efficiently without blocking the main thread.

Question 3 - What is the output?
Explanation:
The output will be:

Start
running..
End

This is because the `blockMainThread()` function blocks the main thread for approximately 3 seconds before logging `"running.."` and then `"End"` sequentially.

Question 4 - What is the output?
Explanation:
The output will be:

d Runs
c
b
a

This is because the function `d()` is executed synchronously, while the `setTimeout` functions are added to the task queue and executed in the order of their specified timeouts.

Question 5 - What's the output?
Explanation:
The output will be:

3
3
3

This happens because in both versions of the `a()` function, the `setTimeout` functions capture the final value of `i` (which is 3 after the loop completes) due to closure, resulting in all three `console.log(i)` statements logging 3 after waiting for their respective timeouts.

Question 6 - What's the Output?
Explanation:
The output will be:

e Runs
b Runs
c Runs
d Runs

This occurs because promises and their respective `then` handlers are executed as microtasks, following the microtask queue's order, and then the `setTimeout` callback runs as a macrotask after the microtasks are completed.

Question 7 - What's the Output?
Explanation:
The output will be:

Start
resolved: 1.3

This happens because the `pause` function returns a promise that resolves after 1000 milliseconds, and the `then` handler calculates the time difference between the start and end times, logging `"resolved: 1.3"` after approximately 1.3 seconds.
