### **Question number 1**

What are microtasks? What is a microtask queue? What is their role in Promises and how are they different from callbacks?

### **Ans:**

### **Microtasks:** IT’s a function that is executed after the function or program which created it exits and the JS execution stack is empty, but before returning control to the event loop. Micro-tasks are often scheduled for things that are required to be completed immediately after the execution of the current script.

The event loop does not move to the next task outside of the micro-task queue until all the tasks inside the micro-task queue are completed.

This means the micro-task queue has a higher priority.

Once all the tasks inside the micro-task queue are finished, only then does the event loop shift back to the macro-task queue. The primary reason for prioritizing the micro-task queue is to improve the user experience. The micro-task queue is processed after callbacks given that any other JavaScript is not under mid-execution. Micro-tasks include mutation observer callbacks as well as promise callbacks.

In such a case wherein new micro-tasks are being added to the queue, these additional micro-tasks are added at the end of the micro-queue and these are also processed. This is because the event loop will keep on calling micro-tasks until there are no more micro-tasks left in the queue, even if new tasks keep getting added. Another important reason for using micro-tasks is to ensure consistent ordering of tasks as well as simultaneously reducing the risk of delays caused by users.

### 

### **QueueMicrotaks**: If we’d like to execute a function asynchronously (after the current code), but before changes are rendered or new events are handled, we can schedule it with **QueueMicrotaks**.

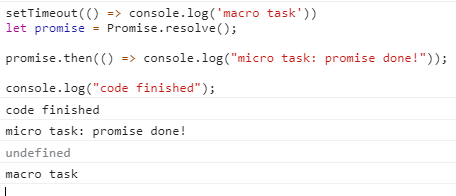
queueMicrotask(() => {

// Code to be run inside the micro-task

});

### **Promises:** Promise handlers .then/.catch/.finally are always asynchronous.

Even when a Promise is immediately resolved, the code on the lines *below* .then/.catch/.finally will still execute before these handlers.



you see **code finished** the first

promise resolved and consoles **micro task: promise done!** Second

And finally macro task console **macro task**

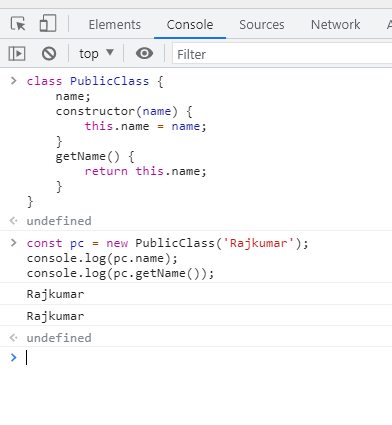
### **Question number 2**

**Explain with examples how private, protected variables can be implemented in classes and how can they be used in subclasses?**

When we work on object-oriented language, we must know the importance of internal and external interfaces. Internal means methods and variables(properties) can only be accessible by the class itself not from outside. In contrast, the external interface has methods and properties that can be accessed from outside.

Three keywords we hear every time we talk about accessibility in class

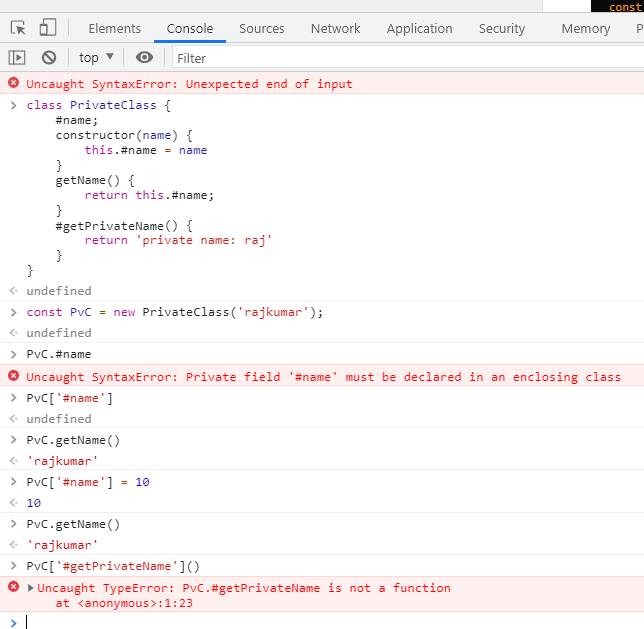
**Public**: These members of the class and available to everyone that can access the (owner) class instance.

****

In The Above class, the name is a public variable and getName is a public method

So, can be accessed from outside the class with the class instance

**Private**: These members are only accessible within the class that instantiated the object.



In The Above class, the **#name** is a **private** variable, **getName** is a **public** method and **#getPrivateName** is a **private** method

So, we can access name only by calling the getName method, and also as **#getPrivateName** function is **private** So,we can’t access it from outside the class using instance

**Protected**: This keyword allows a little more access than private members but a lot less than the public. A protected member is accessible within the class (similar to private) and any object that inherits from it. A protected value is shared across all layers of the prototype chain. It is not accessible by anybody else.



In the able example ProtectedClass has one protected variable \_name which can only be accessible with in that class and and ProtectedChild inherited class  
And ProtectedClass has getter to read the data but no setter to update the data