### **Question number 1**

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

Let us understand the above using a cope snippet.

| console.log(“Start”)  setTimeout(function cbt(){  console.log(“Inside Timeout Callback”)  },5000)  fetch(<https://youtube.com>)  .then(function cbf(){  console.log(“Inside Fetch Callback”)  }  )  console.log(“End”) |
| --- |

So as we know, Javascript has a single call stack. When the code starts executing, a global execution context is created inside the call stack. Now let's go line by line:

1. The console.log() goes inside the call stack and is executed immediately and “Start” is printed in the console.
2. Now the code discovers the setTimeout function (Interesting Fact: It is a feature of browsers Web API and not a part of javascript). The callback function is then registered inside the Web API environment and the timer for 5000ms starts.
3. The code then reaches the fetch which is also a part of Web API and so the callback function cbf is similarly stored in the Web API environment . Now suppose the servers at youtube are really fast and we get the data super soon. The function cbf is now ready to be executed but as suspected it wont enter the callback queue and would rather enter the microtask queue which is of higher priority than the callback queue .This happened because cbf is a promise callback function which comes under mutation observer callback and thus is a microtask. A microtask similar to other callback functions is executed only once the call stack is empty, the difference lies in the fact since microtasks are stored in the microtasks queue the event loop first checks this queue once the call stack is empty. Only after all the callbacks from the microtasks queue are executed does the event loop check the callback queue. This could lead to a situation known as starvation where suppose the one microtask creates another microtask which inturn creates another and this goes on , in such a situation the callback queue does not get a chance to get executed.
4. Now the code reaches to the end and prints “End” in the console.
5. Now since the call stack is empty, the event loop first checks the microtask queue and finds cbf and executes it .”Inside Fetch CallBack” is now printed in the console.
6. Since the microtask queue is empty now, the event loop check the callback queue and executes the cbt function.

The microtask play an important role in promise and they symbolize that the promise is now ready. Once the promise is ready, the handlers like then , catch,etc. are put into the queue.

### **Question number 2**

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

Private and protected are access modifiers that specify the accessibility of variables.

Private variables can only be accessed within a class. It can not be accessed from outside the class.

| class First{  private int data=40;  }    public class Second{  public static void main(String args[]){  First obj=new First();  System.out.println(obj.data);//Compile Time Error  } |
| --- |

The code above gives a compile time error as a private variable is being accessed outside of the class that it has been declared.

Protected variables allow access of the variable within the package or outside of the package through child class. It can not be accessed outside the package without a child class.

| //save by First.java  package pack;  class First{  void msg(){  System.out.println("Hello");  }  }  //save by Second.java  package mypack;  import pack.\*;  class Second{  public static void main(String args[]){  First obj = new First();//Compile Time Error  obj.msg();//Compile Time Error  }  } |
| --- |

The code above gives compile time error since the protected method is being accessed outside of the package when Second class is not a subclass of First class.