**YouKraft Assignment**

**Questions:**

Why Tasks queue are needed?

Ans: Let us take an example of a setTimeOut function,

**setTimeOut**(callback, ms) when this method is invoked it must run the below two steps.

1. Wait “ms” milliseconds.
2. Invoke callback

This is running on main thread, so lets say that this function has 200millisecond as waiting time, since it is running on main thread it will block other functions too, hence we Run the above two steps in parallel but when we do this we create another set of problem as we are now calling back from something other than the main thread( in simple words we will end up with lots of java script running in parallel or get caught in “Race condition”.

Hence a better approach is to “queue a task” (eg when we fetch something how did we get the response into our javascript? It just queues a task)

**What are microtasks?**

A microtask is a short function which is executed after the function or program which created it exits and only if the [JavaScript execution stack](https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop#stack) is empty

JavaScript [promises](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) and the [Mutation Observer API](https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver) both use the microtask queue to run their callbacks.

# **Difference Between Callbacks and Promises**

A key difference between the two is when using the callback approach, we’d normally just pass a callback into a function that would then get called upon completion in order to get the result of something. In promises, however, we attach callbacks on the returned promise object

Callbacks:

function getMoneyBack(money, callback) {

    if (typeof money !== 'number') {

      callback(null, new Error('money is not a number'))

    } else {

      callback(money)

    }

  }

  const money = getMoneyBack(1200)

  console.log(money)

Promises:

function getMoneyBack(money) {

    return new Promise((resolve, reject) => {

      if (typeof money !== 'number') {

        reject(new Error('money is not a number'))

      } else {

        resolve(money)

      }

    })

  }

  getMoneyBack(1200).then((money) => {

    console.log(money)

  })

**2.Explain how private, protected variable can be implemented in a classes and how can they be used in subclasses?**

1. Private: These members are only accessible within the class that instantiated the object.
2. 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.

**Example**

Private variable using closure:

The protected methods and properties can be accessed by applying the protected key (the cKey variable).