1. html、css
2. Js
   1. 执行上下文和作用域（个人总结）：
      1. 执行上下文分为全局执行上下文和函数执行上下文，每个上下文都对应了一个变量对象，这个对象中包含了当前上下文的变量信息以及当前上下文的作用域链
   2. Promise手写
      1. 基础：<https://es6.ruanyifeng.com/#docs/promise>
      2. 手写A+（这个教程循序渐进地实现了Promise，从基本的Promise构造函数和then方法、再到then方法的异步调用再到then方法的链式调用）：[https://juejin.cn/post/6945319439772434469 （EventLoop和宏、微任务）](https://juejin.cn/post/6945319439772434469（EventLoop和宏、微任务）)
      3. Promise面试题：<https://juejin.cn/post/6844904077537574919>
      4. 自己总结的几点知识点：
         1. new Promise()内部的函数是同步函数
         2. Eventloop机制，先执行当前宏任务，再检查微任务并清空后，重新渲染，执行下一轮宏任务（**宏任务**：scripts、IO、Render、定时器；**微任务**：Promise.prototype.then，Promise为基础开发的其他api）
         3. then。。。catch方法具有异常值穿透的特性(即传入的不是函数就是异常值)，并且catch方法也会返回一个promise对象
         4. then方法返回的结果如果是值则返回完成态，如果是抛出错误则返回错误态，如果是返回promise对象，则由promise对象的状态决定。
         5. finally():
            1. .finally()方法不管Promise对象最后的状态如何都会执行
            2. .finally()方法的回调函数不接受任何的参数，也就是说你在.finally()函数中是没法知道Promise最终的状态是resolved还是rejected的
            3. 它最终返回的默认会是一个上一次的Promise对象值，不过如果抛出的是一个异常则返回异常的Promise对象。
         6. all和race：都并行执行一组异步任务，all在所有异步任务都完成后进行回调，而race保留第一个异步任务完成后的结果，其它的执行完后被丢弃。
         7. async和await：
            1. 紧跟着await后面的语句相当于放到了new Promise中，下一行及之后的语句相当于放在Promise.then中。
            2. 在async中，如果 await后面的内容是一个异常或者错误的话，****则终止错误结果，不会继续向下执行****