Առաջադրանք 1. Resolve և Reject մեթոդներ

Promise-ին կարող ենք նաև ֆունկցիայի return-ի միջոցով վերադարձնել, օրինակ՝

```
function tviStugum(number) {
    return new Promise((resolve, reject) => {
        if (number > 10) {
            resolve("Թիվը տասից մեծ է:");
        } else {
            reject("Թիվը տասից փոքր է:");
        }
    });
}

tviStugum(4)
    .then((message) => console.log(message))
    .catch((error) => console.error(error));
```

Օգտագործելով նույն տրամաբանությունը գրել **Promise**, որը ստուգում է number թիվը: Եթե այդ թիվը զույգ է, ապա կանչում է **resolve**(), հակառակ դեպքում **reject**():

Առաջադրանք 2. Ասինխրոն գործողություններ Promise-ի հետ:

Promise-ի մեջ կարող ենք օգտագործել **setTimeout** որը աշխատելուց հետո կարող է կանչել **resolve()**:

```
function asinxronFunctia(miliseconds) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Երկու վարկյան հետո աշխատեցի:");
          }, miliseconds);
    });
```

```
}
asinxronFunctia(2000)
    .then((message) => console.log(message))
    .catch((error) => console.error(error));
```

Օգտագործելով նույն մոտեցումը, ձևափոխեք ֆունկցիան այնպես, որ եթե ֆունկցիային փոխանցվի **1000** միլիվարկյանից փոքր ժամանակ կամ **3000** միլիվարկյանից մեծ ժամանակ, կանչվի **reject**(), որը կփոխանցի սխալ ժամանակի մասին հաղորդագրություն **catch**-ի **error** փոփոխականին և կտպի քանսոլում հողորդագրություն:

Առաջադրանք 3. Շղթայական ֆունկցիաների կանչ:

Վերադարձվող **Promise**-ների օգնությամբ կարող ենք սինխրոնիզացնել JavaScript կոդի աշխատանքը, պարտադրելով կոմպիլիատորին ֆունկցիաները իրականացնել հաջորդաբար, մեկը մյուսի աշխատանքի ավարտից հետո:

```
function step1() {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log("քայլ 1");
            resolve();
        }, 1000);
    });
}

function step2() {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log("քայլ 2");
            resolve();
```

```
        }, 500);
    });
}

step1().then(step2).then(() => console.log("Ավարտվեց!"));
```

Քանի որ կարելի է ավելացնել ցանկացած քանակի **.then()** մեկը մյուսից հետո, օգտագործելով վերոգրյալ օրինակը ընդլայնել քայլերի քանակը՝ ավելացնել Քայլ 3, որը պետք է աշխատի 1500 միլիվարկյան հետո և Քայլ 4, որը պետք է աշխատի 2000 միլիվարկյան հետո:

Առաջադրանք 4. Պատահականություն և Promise:

Ստեղծել ֆունկցիա, որը գեներացնում է պատահական թիվ **Math.random()** հատկությամբ: Անհրաժեշտ է կանչել **resolve()** եթե պատակահան թիվը մեծ է 0,5 ից և **reject()** հակառակ դեպքում: