WEB 講義 参考プリント No.1

・setTimeout/setInterval

Ex1

```
setTimeout(()=>{
    console.log("hello world")
},1000)

setInterval(()=>{
    console.log("hello world(繰り返し)")
})
```

(1000 ミリ秒= 1 秒)

setTimeout(関数,ミリ秒)でミリ秒後に関数を実行する。

setInterval(関数,ミリ秒)でミリ秒ごとに実行する。

・async/await 非同期処理

Ex2 promise

```
const promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("hello world")
    }, 1000)
})

promise.then(data => {
    console.log("log:" + data)
}).catch(e=>{
    console.log("エラーが出ました。"+e)
})
```

```
>>> "log:hello world"
>>> "hello world" (一秒後)
```

Promise とは非同期の処理をするためにある。

Resolve は解決、reject はエラーを出すための関数。

今回ではこの promise は Promise<string>型になる。string を後で返す、という意味の関数。

Then はこの後に実行する。Catch はエラーが出たときの処理。

Ex3 1 秒待つコード

```
async function main() {
    console.log("1")
    await new Promise((resolve) => {
```

```
        setTimeout(resolve, 1000)
    })
    console.log("2")
}
main()
```

```
>> "1"
>> "2"(1秒後)
```

Promise を待機した後に実行するには、async を関数の前につけ、非同期関数にする必要がある。

Ex4 非同期関数

```
async function getStringInAMinute(){
    await new Promise((resolve) => {
        setTimeout(resolve, 1000)
    })
    return "hello world"
}
const result= await getStringInAMinute()
```

非同期関数に return をつけて返り値をつけるようにすると、Promise と同様に await を用いることができる。(今回は await が非同期関数の中で実行されていないため、エラーが出る。Ex3 にように main 非同期関数内で実行すること。)

Ex5 axios で post,get 通信をする

```
const axios=require("axios")
async function main(){
    const response=axios.get("https://google.com")
    // const response=axios.post("https://google.com")
    // post も実行できる。
    const data=response.data
    console.log(data)
}
main()
```

axios というパッケージを用いてコードを実行する。
post 通信: データを URL に渡して、データをもらう。
get 通信: 単に URL からデータを得る。普段サイトをみるときの html は get 通信で取得されている。