Single Threaded Programming

JavaScript

Blocking

Synchronous Execution

Asynchronous WebAPIs

- To achieve asynchronous behavior through
  - ✓Event Loop
  - ✓Callbacks
  - ✓Promises
  - ✓Async and Await

Event Loop

```
function first() {
  console.log(1)
}

function second() {
  console.log(2)
}

function third() {
  console.log(3)
}
```

```
//order of function calls
first()
second()
third()


Output:
1
2
3
```

Event Loop

```
//one of them contains asynchronous code
function first() {
  console.log(1)
}

function second() {
  setTimeout(() => {
    console.log(2)
  }, 0)
}

function third() {
  console.log(3)
}
```

```
//order of function calls
first()
second()
third()


Output:
1
3
2
```

Callback Functions

```
// A function
function fn() {
  console.log('Just a function')
}

// A function that takes another function as an argument
function higherOrderFunction(callback) {
    callback()
}

higherOrderFunction(fn)
```

Nested Callbacks

```
function pyramidOfDoom() {
  setTimeout(() => {
    console.log(1)
    setTimeout(() => {
      console.log(2)
      setTimeout(() => {
        console.log(3)
      }, 500)
    }, 2000)
  }, 1000)
}
```

setTimeout makes the function to wait before executing.

Until then nothing gets executed

```
Output:
1
2
3
```

Promise

A promise represents the completion of an asynchronous function

```
// Initialize a promise
const promise = new Promise((resolve,
reject) => {})
```

Promise

A promise can have three possible states: pending, fulfilled, and rejected.

- Pending - Initial state before being resolved or rejected
- Fulfilled - Successful operation, promise has resolved
- Rejected - Failed operation, promise has rejected

Promise

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Resolving an asynchronous
request!'), 2000)
})

// Log the result
promise.then((response) => {
  console.log(response)
})
```

Error Handling

```
function getUsers(onSuccess) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (onSuccess) {
        resolve([
          {id: 1, name: 'Jerry'},
          {id: 2, name: 'Elaine'},
          {id: 3, name: 'George'},
        ])
      } else {
        reject('Failed to fetch data!')
      }
    }, 1000)
  })
}
```

```
getUsers(false)
  .then((response) => {
    console.log(response)
  })
  .catch((error) => {
    console.error(error)
  })
```

Async and Await

- The async and await keywords in JavaScript are used to make asynchronous programming easy, by introducing coroutines.

- The await keyword is a special command which tells JavaScript to stop the execution of the current function until a Promise resolves, and then return the promise's value.

- Every time we need to run an async function, we need to await on it.

Async and Await

```
function who() {
    return new Promise(resolve => {
      setTimeout(() => {
        resolve('You\'re');
      }, 200);
    });
  }
  function what() {
    return new Promise(resolve => {
      setTimeout(() => {
        resolve('learning');
      }, 300);
    });
  }
  function where() {
    return new Promise(resolve => {
      setTimeout(() => {
        resolve('in the classroom');
      }, 500);
    });
  }
```

```
  async function msg() {
    const a = await who();
    const b = await what();
    const c = await where();

    console.log(`${ a } ${ b } ${ c }`);
  }

  msg();

// You're learning in the classroom <-- after
1 second
```

Async and Await

```
function sleep(ms) {
    return new Promise((resolve) => setTimeout(resolve, ms));
}

function sumAsync(x, y) {
    return new Promise((resolve, reject) => {
        sleep(500).then(() => {
            resolve(x + y);
        });
    });
}

sumAsync(5, 7).then((result) => {
    console.log("The result of the addition is:", result);
});
```

Async and Await

```
function sleep(ms) {
    return new Promise((resolve) => setTimeout(resolve, ms));
}

async function sumAsync(x, y) {
    // this code waits here for 500 milliseconds
    await sleep(500);
    // done waiting. let's calculate and return the value
    return x+y;
}

// sumAsync is an async function, which means it returns a Promise.
sumAsync(5, 7).then((result) => {
    console.log("The result of the addition is:", result);
});
```