4주차

소프트웨어 시스템 설계 및 개발

2024.1학기

CONTENTS

1. Node.js 기초 - Node.js 교과서(길벗)
2. 예제
3. 실습

비동기 예시

```
const fs = require('fs');

// 비동기적으로 파일 읽기
fs.readFile('example.txt', 'utf8', (err, data) => {
    if (err) {
        console.error(err);
        return;
    }
    console.log('파일 내용:', data);
});

console.log('파일을 읽는 중...');
```

```
파일을 읽는 중...
파일 내용: [example.txt 파일의 내용]
```

프로미스 예시

```
let promise = new Promise(function(resolve, reject) {
    // 비동기 작업을 수행한다.
    if (ture or false) {
        resolve("성공 데이터");
    } else {
        reject("실패 데이터");
    }
});

promise.then(
    result => console.log(result), // 성공(이행) 시 실행
    error => console.log(error) // 실패(거부) 시 실행
);
```

- 프로미스는 JavaScript에서 비동기 작업의 최종 완료(또는 실패)와 그 결과값을 나타내는 객체로 세 가지 상태를 가짐
  - 대기(Pending): 초기 상태, 성공 또는 실패로 넘어갈 수 있음
  - 이행(Fulfilled): 연산이 성공적으로 완료됨
  - 거부(Rejected): 연산이 실패함

Async/Await 예시

```
async function fetchData() {
    try {
        let response = await fetch('url'); // fetch 요청이 완료될 때까지 기다린다.
        let data = await response.json(); // 응답을 JSON 형태로 변환할 때까지 기다린다.
        console.log(data); // 변환된 데이터를 출력한다.
    } catch (error) {
        console.error(error); // 오류가 발생하면 catch 블록이 실행된다.
    }
}

fetchData();
```

- **async로 함수를 정의하면 해당 함수는 항상 프로미스를 반환**
- **await 키워드는 async 함수 내에서만 사용되며, 프로미스가 처리될 때까지 함수 실행을 일시 중지하고, 결과 값을 반환**

프로미스와 async/await

- **프로미스**
  - 비동기 작업을 나타내는 객체로, 그 작업이 완료되면 하나의 값으로 결과를 반환
  - 비동기 작업이 성공(resolve)했을 때와 실패(reject)했을 때를 처리하기 위해 .then(), .catch(), .finally()와 같은 메서드를 체인 형태로 사용
- **async/await**
  - async 함수는 항상 Promise를 반환하며, 함수 내부에서 비동기 작업의 완료를 기다리기 위해 await 키워드를 사용
  - await 키워드는 Promise의 완료를 기다리며, Promise가 resolve될 때까지 함수 실행을 일시 중단하고 resolve된 값은 그 다음 라인의 코드로 반환
  - async/await를 사용하면 비동기 코드를 동기 코드처럼 직관적으로 작성할 수 있으며, 복잡한 프로미스 체인을 피할 수 있음
  - 오류 처리는 try/catch 구문을 사용하여 수행할 수 있으며, 이는 동기 코드에서 오류를 처리하는 방법과 유사함
- **차이점**
  - **가독성과 간결성:** async/await를 사용하면 비동기 코드를 동기 코드처럼 읽고 쓸 수 있어 가독성이 더 좋고, 코드가 간결해짐
  - **오류 처리:** async/await는 try/catch를 사용하여 오류를 처리하는 반면, Promise는 .catch() 메서드를 사용
  - **코드 스타일:** Promise는 체인 형태의 비동기 작업 처리를 가능하게 하고, async/await는 더 선언적인 방식을 제공
  - **사용 상황:** 단일 비동기 작업이나 간단한 작업 순서에는 Promise만으로 충분할 수 있지만, 복잡한 비동기 로직, 특히 여러 비동기 작업의 결과가 서로 의존하는 경우 async/await가 더 적합할 수 있음

예제 1

- 1을 입력하면 'hello', 2를 입력하면 'world'를 출력하는 프로그램

```
const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});

readline.question('숫자를 입력해주세요 (1 또는 2): ', input => {
    if (input === '1') {
        console.log('hello');
    } else if (input === '2') {
        console.log('world');
    } else {
        console.log('1 또는 2를 입력해주세요.');
    }

    readline.close();
});
```

예제 2

- 텍스트 입력 후 인덱스를 지정하고 인덱스로부터 앞, 또는 뒤를 지우는 프로그램

```
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function askText() {
  rl.question('영어로 텍스트를 입력해주세요: ', (text) => {
    askIndex(text);
  });
}
```

예제 2

```
function askIndex(text) {
  rl.question(`인덱스를 입력해주세요 (0에서 ${text.length - 1} 사이): `, (index) => {
    if (index >= 0 && index < text.length) {
      askDirection(text, index);
    } else {
      console.log('유효하지 않은 인덱스입니다. 다시 입력해주세요.');
      askIndex(text);
    }
  });
}

function askDirection(text, index) {
  rl.question('인덱스부터 앞을 지울지 뒤를 지울지 선택해주세요. 앞 ("<"), 뒤 (">"): ', (direction) => {
    if (direction === '<') {
      console.log('결과:', text.substring(index));
      rl.close();
    } else if (direction === '>') {
      console.log('결과:', text.substring(0, parseInt(index) + 1));
      rl.close();
    } else {
      console.log('"<" 또는 ">"를 입력해주세요.');
      askDirection(text, index);
    }
  });
}

askText();
```

예제 3

- 사용자로부터 입력 받은 숫자만큼 주사위를 굴리고, 주사위가 나온 횟수와 확률을 보여주는 프로그램

```
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.question('주사위를 굴릴 횟수를 입력해주세요 (100~1000000): ', (input) => {
  const rolls = parseInt(input);
  if (rolls < 100 || rolls > 1000000 || isNaN(rolls)) {
    console.log('입력이 유효하지 않습니다. 프로그램을 종료합니다.');
    rl.close();
    return;
  }

  const results = rollDice(rolls);
  displayResults(results, rolls);
  rl.close();
});
```

예제 3

```
function rollDice(rolls) {
  const results = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0};
  for (let i = 0; i < rolls; i++) {
    const result = Math.floor(Math.random() * 6) + 1;
    results[result]++;
  }
  return results;
}

function displayResults(results, totalRolls) {
  console.log('주사위 굴리기 결과:');
  for (const [side, count] of Object.entries(results)) {
    const probability = ((count / totalRolls) * 100).toFixed(2); // 확률을 백분율로 변환
    console.log(`${side}이(가) 나온 횟수: ${count}, 확률: ${probability}%`);
  }
}
```

개인 실습

- 사용자에게 트럼프 카드를 5장 나눠주고, 5장 중에 포커 조합이 있는지 검색하는 프로그램
- 단 여기서는 페어, 투페어, 쓰리 오브 카인드, 풀하우스, 포카드만 체크함

```
function generateDeck() {
    const suits = ['♠', '♥', '♦', '♣'];
    const ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A'];
    const deck = [];

    for (let suit of suits) {
        for (let rank of ranks) {
            deck.push({ rank, suit });
        }
    }

    return deck;
}

function shuffleDeck(deck) {
    for (let i = deck.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [deck[i], deck[j]] = [deck[j], deck[i]];
    }
}

function dealCards(deck, numCards) {
    return deck.slice(0, numCards);
}
```

개인 실습

```
function checkHand(hand) {
    const ranks = hand.map(card => card.rank);
    const counts = ranks.reduce((acc, rank) => {
        acc[rank] = (acc[rank] || 0) + 1;
        return acc;
    }, {});

    const duplicates = Object.values(counts).reduce((acc, count) => {
      if (count > 1) acc.push(count);
      return acc;
    }, []);

    // 스트레이트 플러쉬
    if (duplicates.includes(4)) return "Four of a Kind";
    if (duplicates.includes(3) && duplicates.includes(2)) return "Full House";
    // 플러쉬
    // 스트레이트
    if (duplicates.includes(3)) return "Three of a Kind";
    if (duplicates.length === 2) return "Two Pair";
    if (duplicates.includes(2)) return "One Pair";
    return "No Combination";
}

function main() {
    const deck = generateDeck();
    shuffleDeck(deck);
    const hand = dealCards(deck, 5);

    console.log("Your hand:", hand.map(card => `${card.suit}${card.rank}`).join(', '));
    console.log("Combination:", checkHand(hand));
}

main();
```

개인 실습

- 예제1, 2, 3의 소스코드를 참고하여 사용자에게 입력 받은 횟수만큼 카드 게임을 반복하여 플레이하게 수정하시오. (난이도1)
- 입력 받은 횟수만큼 반복한 후 어떤 포커 조합이 몇 번 나왔는지 출력하도록 수정하시오. (난이도2)
- 스트레이트, 플러쉬, 스트레이트 플러쉬 등 다른 포커 조합의 검사식을 추가하시오. (도전과제)