**1. FLEX레이아웃과 GRID 레이아웃을 설명하고**

**예제를 만드시오.**

**Flex** - 한 방향 레이아웃 시스템 (1차원)

**Grid** - 두 방향(가로-세로) 레이아웃 시스템 (2차원)

* Grid Container : 부모 요소인 div.container
* Grid Item : 자식 요소인 div.item

<!DOCTYPE html>

<html lang="ko">

<head>

  <meta charset="utf-8">

  <title>Grid</title>

    <link rel="stylesheet" media="all" href="" />

  <style>

    .grid-container {

      width: 100%;

      display: grid;

      grid-template-rows: 70px 500px 70px;

      grid-template-columns: repeat(3, 1fr);

      gap: 10px

    }

    .grid-item{

      color: #ffffff;

      background: #333;

      display: flex;

      justify-content: center;

      align-items: center;

    }

    .grid-item.holy {

      grid-column: 1 / -1;

    }

    .grid-item.footer {

      grid-column: 1 / -1;

    }

  </style>

</head>

<body>

  <div class="grid-container">

    <div class="grid-item holy">Holy Grail Layout</div>

    <div class="grid-item nav">Nav</div>

    <div class="grid-item main">Main Content</div>

    <div class="grid-item sidebar">Sidebar</div>

    <div class="grid-item footer">Footer</div>

  </div>

</body>

<script>

'use strict';

</script>

</html>

**2. ES6문법을 10가지이상 설명하고 각 예제를 만드시오.**

**1. Let**

* let: 변수
* let, const는 block scope( {}안에서만 유효 범위(scope)를 가져서, {}외부에서는 읽히지 않는 성질 )를 갖는다.

function fruit () {

  if(true){

    var apple = 1;

    let banana=7;

  }

  console.log(apple, banana)

  // Uncaught ReferenceError: banana is not defined

}

fruit()

**2. const**

- const: 변경되지 않는 상수

- let, const는 block scope( {}안에서만 유효 범위(scope)를 가져서, {}외부에서는 읽히지 않는 성질 )를 갖는다.

const melon = "melon"

melon = "apple"

// Uncaught TypeError: Assignment to constant variable.

**3. arrow function**

- parameta가 하나일 때 괄호 생략이 가능하다. (여러 개일 경우는 ()로 감싸준다.)

- 간단한 표현식만 반환할 때는 return 생략이 가능하다.

// ES5

var fruit = function (eat) {

  return console.log(eat)

}

//  ES6

const fruit = eat => console.log(eat)

fruit('apple')

**4. class**

constructor()

- 속성을 할당하는 method

- class 내부에 하나만 존재할 수 있으며, class를 생성할 때 꼭 들어가야 한다.

- extends 통해서 부모class 상속 가능

class Fruit {

  constructor(prop, flavor) {

    this.prop = 'plant'

    this.flavor = 'awesome'

  }

}

class Apple extends Fruit {

  constructor(prop, flavor){

    super(prop, flavor)

    this.color = 'red'

  }

}

let apple = new Apple()

console.log(apple); // Apple {prop: "plant", flavor: "awesome", color: "red"}

**5. Template literal**

- `` 안에 작성하며, ${}안은 변수, ${}밖은 string 처리된다. 공백, 개행 그대로 표현

// ES5

var apple = '사과';

var flavor = '맛있다';

console.log(apple + "는 과일이고, " + flavor)

// ES6

const apple = '사과';

const flavor = '맛있다';

console.log(`${apple}는 과일이고, ${flavor}`)

**6. module**

- Export, Import 를 이용해 function이나 variables 들을 다른 곳에서 사용할 수 있다.

//  fruit.js

export const fruit = arr => {return true}

// App.js

import { fruit } from "fruit"

console.log(fruit()) // true

**7. Destructuring Assignment**

- 객체 안의 필드를 손쉽게 꺼내어 변수로 대입할 수 있는 문법.

// ES5

var apple = {

  color: 'red',

  shape: 'circle'

}

var color = apple.color;

var shape = apple.shape;

console.log(color, shape); // 'red', 'circle'

// ES6

const apple = {

  color: 'red',

  shape: 'circle'

}

const { color, shape } = apple;

//비구조화 할당

console.log(color, shape); // 'red', 'circle'

**8. 간결한 method**

// ES5

var fruit = {

  prop: 'plant',

  getGrow: function() {

    return 'grow up';

  }

}

console.log(fruit.getGrow());

// ES6

var fruit = {

  prop: 'plant',

  getGrow() {

    return 'grow up';

  }

}

console.log(fruit.getGrow());

**9. spread operator**

- 특정 객체 또는 배열의 값을 다른 객체, 배열로 복제하거나 옮길 때 사용

- 복제할 대상 앞에 ...을 붙여 사용

// ES5

var a = [1,2,3];

var b = [4, 5];

var c = "end";

var d = a.concat(b, c);

console.log(d);

//[1, 2, 3, 4, 5, "end"]

// ES6

const a = [1,2,3];

const b = [4, 5];

const c = "end";

const d = [...a, ...b, c];

console.log(d);

//[1, 2, 3, 4, 5, "end"]

**10. Default Parameter Values**

- 매개변수에 undefined가 아닌 기본값 (default)을 지정해 줄 수 있다.

function apple ( color ='red' , price = 1000 ) {

...

}

**11. Array.find()**

- find() 메서드는 주어진 판별 함수를 만족하는 첫 번째 요소의 값을 반환.

그런 요소가 없다면 undefined를 반환.

const array = [5, 12, 8, 130, 44];

const found = array.find(elem => elem > 10);

console.log(found); // 12

**3. props 와 state는 언제 쓰는가.**

**props**

* 부모 -> 자식 component로 데이터를 전달할 때

**state**

* mutable data를 사용할 때
* component 내에 별도의 상태가 필요할 때

(React.js) React.js의 state 사용방법

<https://medium.com/@yeon22/react-js-react-js%EC%9D%98-state-%EC%82%AC%EC%9A%A9%EB%B0%A9%EB%B2%95-fb47327e685b>