소프트웨어 나눔 축제

AnA - Node.js로 크롤링해보기

자바스크립트

1. 시작전
2. 순서
3. 시작

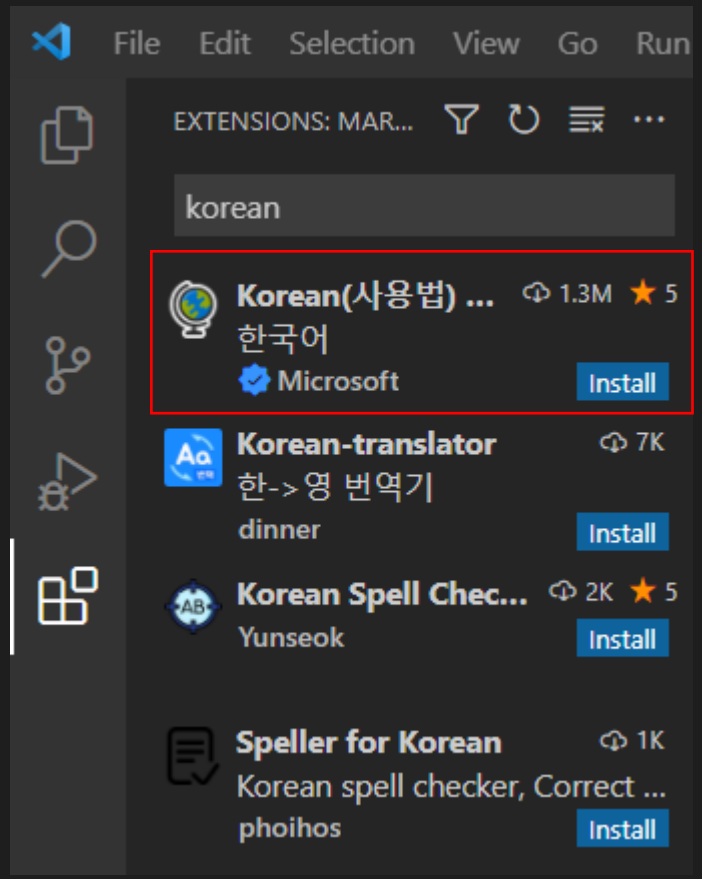
시작전

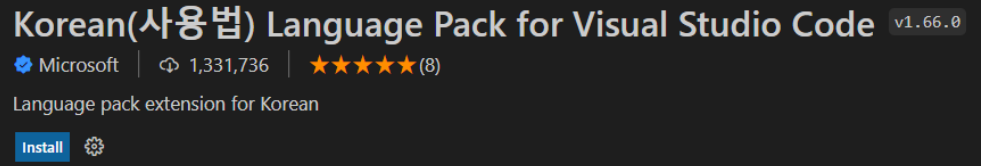
먼저 비주얼 스튜디오 코드 사용방법부터 배워봅시다.

자바스크립트

1. 시작전
2. 순서
3. 시작

시작전 - 비주얼 스튜디오 코드 한국어 패치

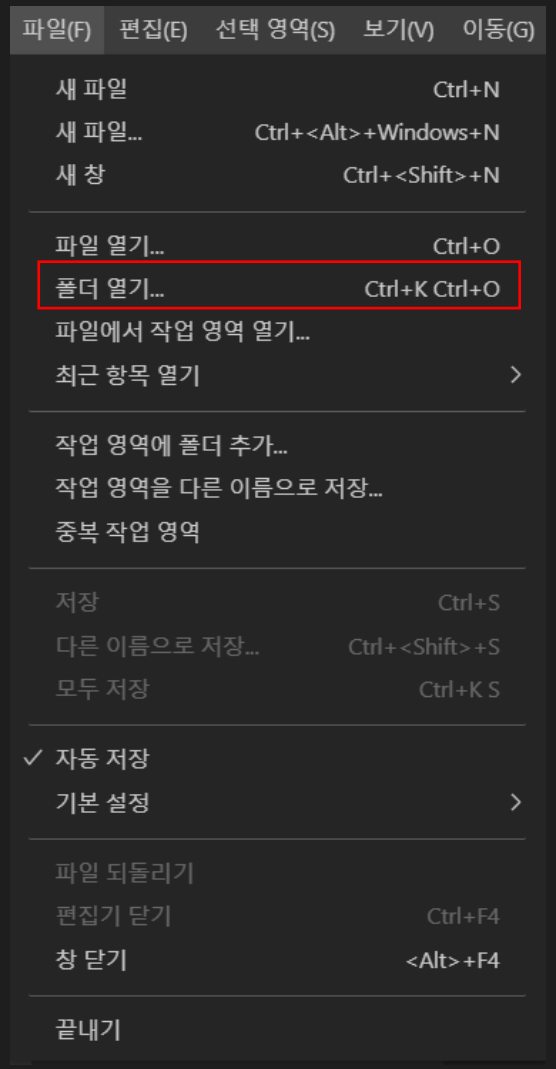


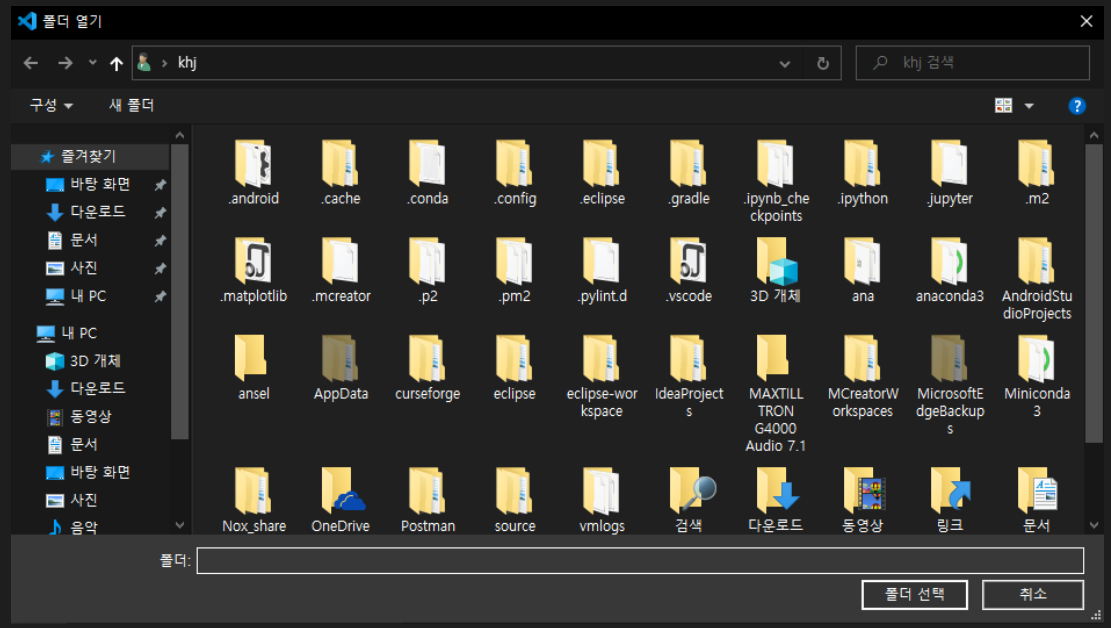


자바스크립트

1. 시작전
2. 순서
3. 시작

시작전 - 비주얼 스튜디오 코드 폴더 열기

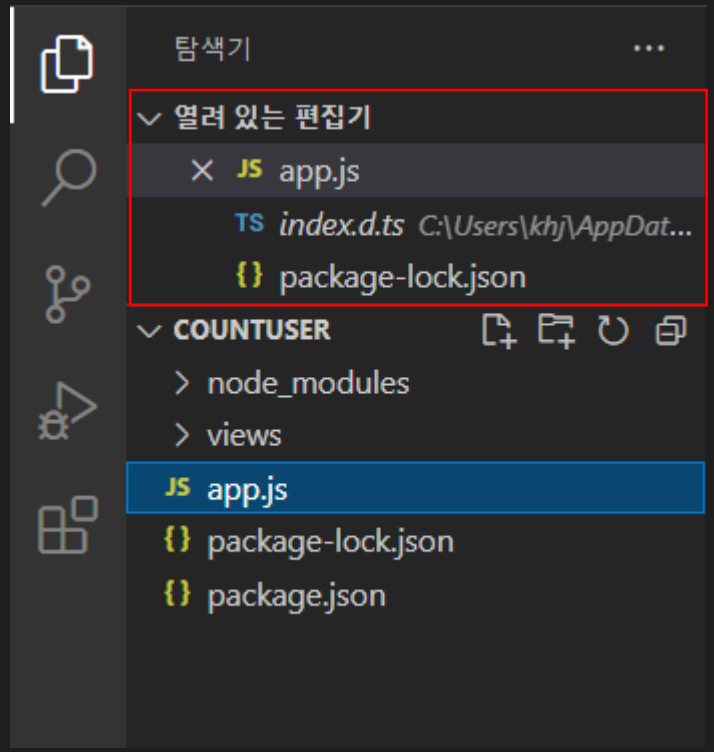


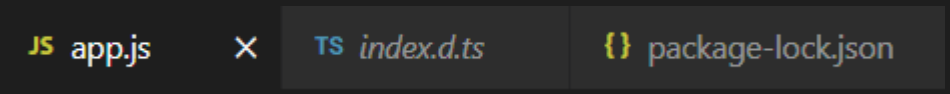


자바스크립트

1. 시작전
2. 순서
3. 시작

시작전 - 비주얼 스튜디오 코드 열려있는 편집기

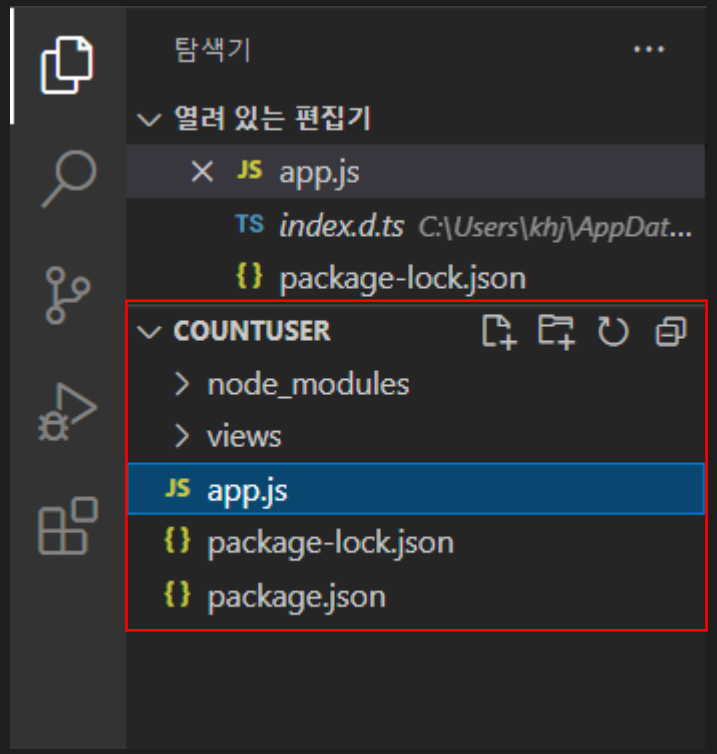




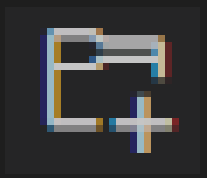
자바스크립트

1. 시작전
2. 순서
3. 시작

시작전 - 비주얼 스튜디오 코드 현재 폴더



새 파일



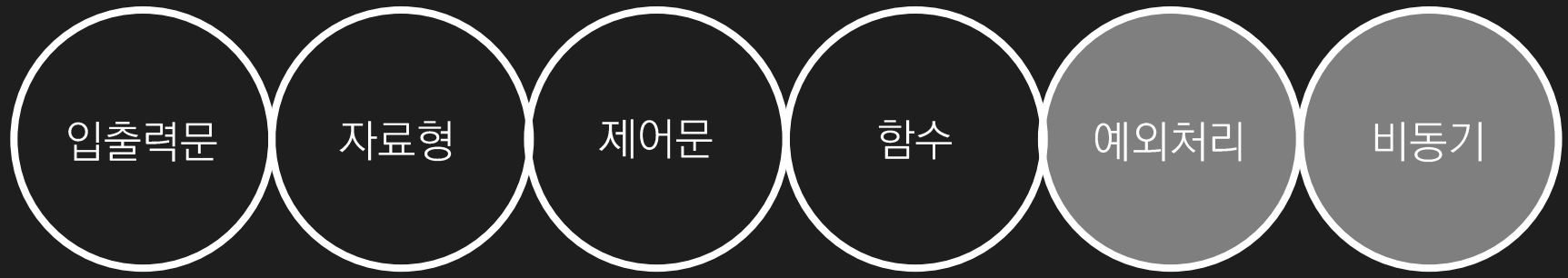
새 폴더

새로 고침

자바스크립트

1. 시작전
2. 순서
3. 시작

자바스크립트 문법 순서



자바스크립트

1. 시작전
2. 순서
3. 시작

자바스크립트란?

“웹페이지에 생동감을 불어넣기 위해”만들어진 프로그래밍 언어입니다.

자바스크립트

1. 시작전
2. 순서
3. 시작

출력함수

console.log를 사용하면 문자열을 출력할 수 있습니다.

```
console.log('2022 SSF AnA');
```

자바스크립트

1. 시작전
2. 순서
3. 시작

변수와 상수

변수(variable)는 데이터를 저장할 때 쓰이는 '이름이 붙은 저장소' 입니다.
자바스크립트에서는 let 키워드를 통해 변수를 생성합니다.

```
let message;
```

자바스크립트

1. 시작전
2. 순서
3. 시작

변수와 상수

상수(constant)는 변화하지 않는 변수를 선언합니다.
const라는 키워드를 통해 사용할 수 있습니다.

```
const message;
```

자바스크립트

1. 시작전
2. 순서
3. 시작

자료형

값은 항상 문자열이나 숫자형 같은 특정한 자료형에 속합니다.
자바스크립트에는 여덟 가지 기본 자료형이 있습니다.

숫자형
BigInt
문자형
불린형
null
undefined
객체형
심볼형

자바스크립트

자료형 - 숫자형

숫자형은 정수와 부동소수점 숫자를 나타냅니다.

```
let number = 123;
number = 1.23;
```

자바스크립트

1. 시작전
2. 순서
3. 시작

자료형 – BigInt

BigInt는 길이에 상관없이 정수를 나타낼 수 있습니다.

```
let bigInt = 1234567891011121314151617181 9;
```

자바스크립트

자료형 – 문자형

문자형은 말 그대로 문자를 저장합니다.

```
let str = '2022 SSF';
```

자바스크립트

1. 시작전
2. 순서
3. 시작

자료형 – 불린형

불린형은 true와 false 두 가지 값밖에 없는 자료형입니다.

```
let isTrue = true;
```

자바스크립트

자료형 – null

null은 존재하지 않거나 알 수 없는 값을 나타내는데 사용합니다.

```
let age = null;
```

자바스크립트

1. 시작전
2. 순서
3. 시작

자료형 – undefined

undefined는 값이 할당되지 않은 상태를 나타내는데 사용합니다.
null과 구분할 필요가 있습니다.

```
let age = null;
```

자바스크립트

자료형 - 객체형과 심볼형

객체형은 다양한 데이터를 담을 수 있습니다.
특이하게도 키와 값 쌍으로 구성된 프로퍼티를 가집니다.

```
let club = {
  name: 'AnA', // 키: 'name', 값: 'AnA'
  age: 13 // 키: 'age', 값: 13
};
```

자바스크립트

1. 시작전
2. 순서
3. 시작

자료형 – 심볼형

심볼형은 패스!

자바스크립트

1. 시작전
2. 순서
3. 시작

조건문

조건에 따라 다른 행동을 취해야 할 때가 있다.

if문

조건문 연산자

자바스크립트

1. 시작전
2. 순서
3. 시작

조건문 - if문

if문은 괄호 안의 조건을 평가하여 코드를 실행하는 제어문입니다.

```
let year = 2022;

if (year == 2022) {
  console.log('2022 SSF!');
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

조건문 - if문

if문에는 else절을 붙일 수 있습니다.
else 뒤에 이어지는 코드 블록은 조건이 거짓일 때 실행됩니다.

```
let year = 2022;

if (year == 2022) {
  console.log('2022 SSF!');
} else {
  console.log('2022?');
}
```

자바스크립트

조건문 - if문

또한 else if절을 붙여 복수 조건을 처리할 수 있습니다.

```
let year = 2022;

if (year == 2022) {
  console.log('2022 SSF!');
} else if (year > 2022) {
  console.log('너무 갔네요!');
} else if (year < 2022) {
  console.log('조금 더!');
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

조건문 - 조건부 연산자 ?

조건에 따라 다른 값을 변수에 할당해줘야 할 때가 있습니다.
조건부 연산자를 사용하면 if문보다 더 짧고 간결하여 변형할 수 있습니다!

```
let result = 2 > 1 ? true : false;
```

자바스크립트

1. 시작전
2. 순서
3. 시작

반복문

반복문을 사용하면 동일한 코드를 여러 번 반복할 수 있습니다.

while

for

자바스크립트

1. 시작전
2. 순서
3. 시작

반복문 - while

간단합니다! 조건이 참이라면 반복하는 문법입니다.

```
time = 13

while(time < 17) {
  console.log('2022 SSF!')
  time++;
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

반복문 - for

for 반복문은 가장 많이 사용되는 반복문입니다.

```
for (let time = 13; time < 17; time++) {
  console.log('2022 SSF');
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

반복문

반복문은 조건이 false가 되면 종료합니다.
그런데 특별한 지시자로 조종할 수도 있습니다.

break

continue

자바스크립트

1. 시작전
2. 순서
3. 시작

반복문

break를 사용하면 언제든 반복문을 빠져나올 수 있습니다.

```
for (let time = 13; time < 17; time++) {
  console.log('2022 SSF’);

  if (time == 15)
    break;
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

반복문

continue는 다음 반복으로 넘어가고 싶을 때 사용할 수 있습니다.

```
for (let time = 13; time < 17; time++) {
  console.log('2022 SSF');
  if (time == 15)
    continue;
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

함수

코드를 재사용해보자!

자바스크립트

1. 시작전
2. 순서
3. 시작

함수 – 선언

먼저 함수를 만들어 봅시다

```
function sayMessage() {
  console.log('2022 SSF');
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

함수 – 호출

이제 사용해 봅시다!

```
sayMessage();
```

자바스크립트

1. 시작전
2. 순서
3. 시작

예외처리

예외처리는 에러가 발생했을 때 합당한 동작을 할 수 있게합니다.

try catch

자바스크립트

예외처리 - try catch

try catch 문법은 try와 catch라는 두 개의 블록으로 구성됩니다.

```
try {
  // 코드
} catch (err) {
  // 에러 핸들링
}
```

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기

비동기란 동시에 일어나지 않는다는 의미입니다.

promise

async

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 - promise

프라미스는 제작코드와 소비코드를 연결해 주는 특별한 객체입니다.
코드를 보며 배워봅시다.

```
new Promise((res, rej) => {
  console.log('출발');
  setTimeout(() => {
    console.log('도착');
    res("활동을 합니다");
  }, 1000);
}).then((res) => {
  console.log(res);
}).finally(() => {
  console.log('집에 갑니다');
})
```

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 – promise

```
new Promise((res, rej) => {
  console.log('출발');
  setTimeout(() => {
    console.log('도착');
    res("활동을 합니다");
  }, 1000);
})
.then((res) => {
  console.log(res);
})
.finally(() => {
  console.log('집에 갑니다');
})
```

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 – promise

```
new Promise((res, rej) => {
  console.log('출발');
  setTimeout(() => {
    console.log('도착');
    res("활동을 합니다");
  }, 1000);
})
.then((res) => {
  console.log(res);
})
.finally(() => {
  console.log('집에 갑니다');
})
```

제작 코드

소비 코드

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 – promise

```
new Promise((res, rej) => {
  console.log('출발');
  setTimeout(() => {
    console.log('도착');
    res("활동을 합니다");
  }, 1000);
})
.then((res) => {
  console.log(res);
})
.finally(() => {
  console.log('집에 갑니다');
})
```

res : 성공적

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 – promise

```
new Promise((res, rej) => {
  console.log('출발');
  setTimeout(() => {
    console.log('도착');
    res("활동을 합니다");
  }, 1000);
})
.then((res) => {
  console.log(res);
})
.finally(() => {
  console.log('집에 갑니다');
})
```

then : 실행 결과

finally : 마지막에 항상

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 - async

async를 사용하면 프라미스를 좀 더 편하게 사용할 수 있습니다.
코드를 보며 배워봅시다.

```
async function func() {
  let promise = new Promise((res, rej) => {
    setTimeout(() => res('값'), 1000)
  });

  let result = await promise; // (*)

  console.log(result);
}

func();
```

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 - async

```
async function func() {
  let promise = new Promise((res, rej) => {
    setTimeout(() => res('값'), 1000)
  });

  let result = await promise; // (*)

  console.log(result);
}

func();
```

자바스크립트

1. 시작전
2. 순서
3. 시작

비동기 - async

```
async function func() {
  let promise = new Promise((res, rej) => {
    setTimeout(() => res('값'), 1000)
  });

  let result = await promise; // (*)

  console.log(result);
}

func();
```

async : function 앞에 위치

await : 비동기 값을 기다림