Ch.4 변수

▼ 🌟🌟메모리

컴퓨터는 CPU를 사용해 연산하고 메모리를 사용해 데이터를 기억한다.

메모리는 데이터를 저장할 수 있는 메모리 셀의 집합체다.

메모리 셀 하나의 크기는 1바이트다.

컴퓨터는 메모리 셀의 크기단위로(즉, 1바이트 단위로) 데이터를 저장하거나 읽어들인다.

각 셀은 고유의 메모리주소를 갖는다.

식별자는 값이 아니라 메모리주소를 기억하고 있다.

▼ 🌟🌟🌟변수

하나의 값을 저장하기 위해 확보한 **메모리 공간 자체** 또는 그 **메모리 공간을 식별하기 위해 붙인 이름.**

변수는 프로그래밍 언어에서 값을 저장하고 참조하는 메커니즘으로 값의 위치를 가리키는 상징적인 이름이다.

변수 이름을 비롯한 **모든 식별자는 실행 컨텍스트에 등록**된다.

실행 컨텍스트는 자바스크립트 엔진이 소스코드를 평가하고 실행하기 위해 필요한 환경을 제공하고 코드의 실행 결과를 실제로 관리하는 영역이다.

자바스크립트 엔진은 실행 컨택스트를 통해 식별자와 스코프를 관리한다.

변수 이름과 변수 값은 실행 컨텍스트 내에 키/값 형식인 객체로 등록되어 관리된다.

▼ 참조

변수에 값을 저장하는 것을 할당이라 하고 변수에 저장된 값을 읽어들이는 것을 참조라 한다.

▼ 변수 선언

값을 저장하기 위한 메모리 공간을 확보하고 변수 이름과 확보된 메모리 공간의 주소를 연결해서 값을 저장할 수 있게 준비하는 것

▼ 🌟자바스크립트 엔진의 변수 선언 2단계

- **선언 단계**: 변수 이름을 등록해서 자바스크립트 엔진에 변수의 존재를 알린다.
- **초기화 단계**: 값을 저장하기 위한 메모리 공간을 확보하고 암묵적으로 undefined를 할당해 초기화한다. (var의 경우)

  +const, let의 경우 선언 후 undefined로 초기화 하지 않는다. 때문에 TDZ가 생긴다.

▼ 🌟🌟🌟**var** vs **const** vs **let**

const와 let으로 선언한 변수는 var와 달리 선언과 동시에 초기화되지 않습니다.

1. **var**:
   - 호이스팅이 발생하여 선언된 변수가 스코프의 최상단으로 끌어올려집니다.
   - 암묵적으로 undefined로 초기화됩니다.
   - 선언 전에 참조할 경우 undefined를 반환합니다.

```
console.log(a); // undefined
var a = 10;
```

2. **let**:
   - 호이스팅이 발생하지만, 선언된 위치에 도달하기 전까지 변수는 "일시적 사각지대(Temporal Dead Zone)"에 존재합니다.
   - 초기화하지 않으면 참조할 수 없습니다.

- 선언 전에 참조할 경우 ReferenceError가 발생합니다.

```
console.log(b); // ReferenceError: Cannot access 'b'
before initialization
let b = 20;
```

3. **const**:
   - let과 유사하게 호이스팅이 발생하지만 "일시적 사각지대(Temporal Dead Zone)"에 존재합니다.
   - 선언 전에 참조할 경우 ReferenceError가 발생합니다.
   - 선언과 동시에 초기화가 반드시 이루어져야 합니다.

     (상수 선언을 지원하여, 선언과 동시에 초기화되고 이후 값이 변경되지 않도록 보장하기 때문)
   - 초기화하지 않으면 SyntaxError가 발생합니다.
   - 재할당이 불가능 합니다.

     ▼ 변수 자체에 대한 재할당이 불가능하다는 뜻이지, 변수에 할당된 값의 내부 상태를 변경할 수 없다는 것을 의미하지는 않습니다. 객체나 배열 같은 참조 타입의 경우에는 변수에 할당된 객체나 배열 자체는 변하지 않지만, 그 내부의 속성이나 요소는 변경할 수 있습니다.

     예시를 통해 살펴보겠습니다:

```
// const로 변수를 선언하고 숫자 값을 할당
const number = 42;
number = 50;  // Error: Assignment to constant
variable

// const로 변수를 선언하고 객체를 할당
const person = { name: 'John', age: 30 };
person.name = 'Jane';  // 가능: 객체 내부 속성 변경
console.log(person.name);  // 출력: 'Jane'

person = { name: 'Doe', age: 25 };  // Error: A
ssignment to constant variable
```

```
// const로 변수를 선언하고 배열을 할당
const numbers = [1, 2, 3];
numbers.push(4);  // 가능: 배열 내부 요소 추가
console.log(numbers);  // 출력: [1, 2, 3, 4]

numbers = [5, 6, 7];  // Error: Assignment to c
onstant variable
```

```
console.log(c); // ReferenceError: Cannot access 'c'
before initialization
const c = 30;
```

+let 키워드로 선언된 변수는 선언과 동시에 초기화하지 않아도 됩니다. 즉, let으로 변수를 선언만 하고 나중에 값을 할당하는 것이 가능합니다.

▼ 🌟🌟🌟변수 호이스팅

변수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징.

변수 선언(선언 단계와 초기화 단계)은 소스코드가 순차적으로 실행되는 런타임 이전 단계에서 먼저 실행된다.

변수 선언 뿐 아니라 var, let, const, function, function*, class 키워드를 사용해서 선언하는 모든 식별자(변수, 함수, 클래스 등)는 호이스팅된다.

▼ function*

function* 키워드는 자바스크립트에서 **제너레이터 함수를 정의**하는 데 사용됩니다.

제너레이터 함수는 일반 함수와 달리 중간에 실행을 멈추고 재개할 수 있는 특별한 함수입니다. 제너레이터 함수는 호출될 때 제너레이터 객체를 반환하며, 이 객체를 통해 함수의 실행을 제어할 수 있습니다.

제너레이터 함수의 주요 특징은 다음과 같습니다:

1. **function* 문법**: 제너레이터 함수는 function* 키워드를 사용하여 정의됩니다.
2. **yield 키워드**: 제너레이터 함수는 yield 키워드를 사용하여 중간 값을 반환하고, 함수의 실행을 멈춥니다.
3. **next 메서드**: 제너레이터 객체의 next 메서드를 호출하여 함수의 실행을 재개할 수 있습니다.

제너레이터 함수의 예:

```
function* countNumbers() {
  let count = 1;
  while (true) {
    yield count++;
  }
}

const counter = countNumbers();

console.log(counter.next().value); // 1
console.log(counter.next().value); // 2
console.log(counter.next().value); // 3
```

위 예제에서 countNumbers 제너레이터 함수는 무한 루프를 사용하여 숫자를 순차적으로 반환합니다. next 메서드를 호출할 때마다 함수는 yield 표현식에서 멈추고, 값을 반환한 후 다음 실행을 기다립니다.

▼ 🌟🌟🌟변수 선언과 값의 할당 실행 시점

변수 선언은 소스코드가 순차적으로 실행되는 시점인 런타임 이전에 먼저 실행.

값의 할당은 런타임에 실행.

▼ 🌟🌟상수

단 한번만 할당할 수 있는 변수 (재할당 불가)

const 키워드를 사용해 선언한 변수는 재할당이 금지된다. 즉, const키워드를 사용하면 상수를 표현할 수 있다.

하지만 const 키워드는 반드시 상수만을 위해 사용하지는 않는다.

▼ 가비지 콜렉터

가비지 콜렉터는 애플리케이션이 할당한 메모리 공간을 주기적으로 검사하여 더이상 사용되지 않는 메모리를 해제하는 기능을 말한다.

더이상 사용되지 않는 메모리란 어떤 식별자도 참조하지 않는 메모리 공간을 의미한다.

▼ 언매니지드 언어 vs 매니지드 언어

p47(e81)

프로그래밍 언어는 메모리 관리 방식에 따라 언매니지드 언어와 매니지드 언어로 분류할 수 있다.

C 언어 같은 언매니지드 언어는 개발자가 명시적으로 메모리를 할당하고 해제하기 위해 malloc( )과 free( ) 같은 저수준 메모리 제어 기능을 제공한다. 언매니지드 언어는 메모리 제어를 개발자가 주도할 수 있으므로 개발자의 역량에 따라 최적의 성능을 확보할 수 있지만 그 반대의 경우 치명적 오류를 생산할 가능성도 있다.

자바스크립트 같은 매니지드 언어는 메모리의 할당 및 해제를 위한 메모리 관리 기능을 언어 차원에서 담당하고 개발자의 직접적인 메모리 제어를 허용하지 않는다. 즉, 개발자가 명시적으로 메모리를 할당하고 해제할 수 없다.

▼ 🌟식별자 네이밍 규칙

- 특수문자를 제외한 문자, 숫자, 언어스코어, 달러기호를 포함할 수 있다.
- 특수문자를 제외한 문자, 언어스코어, 달러기호로 시작해야 한다. 숫자로 시작하는 것은 허용하지 않는다.

- 예약어는 식별자로 사용할 수 없다.

▼ 🌟🌟🌟네이밍 컨벤션

네이밍 컨벤션은 하나 이상의 영어 단어로 구성된 식별자를 만들 때 가독성 좋게 단어를 한눈에 구분하기 위해 규정한 명명 규칙이다.

자바스크립트에서는 일반적으로 변수나 함수이름에는 카멜케이스를 사용하고 생성자함수, 클래스의 이름에는 파스칼 케이스를 사용한다.

- 카멜케이스
- 스네이크 케이스
- 파스칼 케이스
- 헝가리언 케이스