2장 타입

1. 타입이란

   1.1 자료형으로서

   - 값의 종류를 명시하고 메모리를 효율적으로 사용하기 위해 데이터 타입 정의

   1.2 집합으로서

   - 집합 : 값이 가질수있는 유효한 범위의 집합
   - 유효한 값의 밤위를 제한하여 런타임에서 발생할수있는 값에 대한 에러방지

   1.3 정적 타입과 동적 타입

구분	정적 타입 언어	동적 타입 언어
타입 선언	변수 선언 시 타입을 명시해야 함	변수 선언 시 타입을 명시하지 않음
타입 체크	컴파일 타임에 타입 체크	런타임 시 타입 체크
에러 발견	컴파일 단계에서 타입 에러 발견	런타임 시 타입 에러 발견
성능	컴파일러가 최적화할 수 있어 성능 좋음	동적 타입 체크로 인해 성능이 상대적으로 낮음
안정성	컴파일 단계에서 타입 오류를 잡아낼 수 있어 안정적	런타임 오류 가능성이 높음
개발 생산성	타입 정의로 IDE 지원이 우수하지만 초기 개발 속도가 느림	빠른 프로토타입 개발이 가능하지만 디버깅이 어려움
대표 언어	Java, C, C++, C#, Rust, Scala	Python, JavaScript, Ruby, PHP, Perl

   1.4 강타입과 약타입

   - 암묵적 타입 변환
     - 런타임에 타입이 자동으로 변경되는것
     - 암묵적 타입 변환여부에 따라 강타입,약타입으로 구분한다

2. 타입스크립트의 타입시스템

   2.1 타입 애너테이션 방식

- 인수의 타입을 명시적으로 선언하여 어떤 타입값이 저장될것인지를 컴파일러에 직접 알려주는 문법

2.2 구조적 타이핑

- 구조로 타입을 구분하는것

3. 구조적 타이핑
    - 객체가 가지고있는 속성을 바탕으로 타입을 구분하는것

4. 자바스크립트를 닮을 타입스크립트
    - 구조적타이핑과 다르게 명목적 타이핑은 타입의 구조가 아닌 타입의 이름만을 가지고 구별한다
    - 명목적 타이핑은 동일성을 확인하는 과정에서 구조적 타이핑에 비해 조금 더 안전
    - 덕타이핑
        - 타입에 부합하는 변수와 메서드를 가질경우 해당타입에 속하는 것으로 간주

구분	덕 타이핑	구조적 타이핑
정의	객체의 실제 타입보다는 객체가 어떤 메서드와 속성을 가지고 있는지에 초점을 맞춤	객체의 내부 구조(필드, 메서드)가 타입 호환성을 결정함
타입 체크	객체가 필요한 메서드와 속성을 가지고 있는지 여부로 타입을 판단	객체의 구조가 호환되는지 여부로 타입을 판단
대표 언어	Python, Ruby, JavaScript	TypeScript, Scala, Rust
장점	유연성이 높음, 코드가 간결해짐	정적 타입 검사로 안정성 향상
단점	런타임 오류 위험 존재	코드가 다소 복잡해질 수 있음

5. 구조적 타이핑의 결과
    - 구조적 타이핑의 결과로 결과값이 어떤속성을 지닐지 알수없는 경우가 존재한다
    - 이러한 언어의 특징을 극복하기위해 식별자인 유니온 같은 방법을 생성

6. 점진적인 타입 확인

- 컴파일 타입에 타입을 검사하면서 필요에따라 타입선언을 생략하는것
- any 타입
  - 타입스크립트 내 모든 타입의 종류를 포함하는 가상 상위타입

7. 자바스크립트 슈퍼셋으로서의 타입스크립트
   - 기존 자바스크립트 코드에 정적인 타이핑을 추가한것으로 자바스크립트이 상위집합

8. 값 vs 타입
   - 값
     - 프로그램이 처리하기 위해 메모리에 저장하는 모든 데이터
   - 값 공간과 타입공간의 이름은 충돌하지않기 때문에 같은 이름으로 정의가능하다
     - 타입스크립트 문법으로 선언한 내용은 자바스크립트 런타임에 제거
     - 그리하여 값공간과 타입공간은 충돌하지않는다
   - **enum**
     - 열거형을 정의할 수 있음
       - 주로 상수 집합을 표현하는데 유용
     - 런타임에 객체로 변환됨
     - 런타임에 실제 객체로 존재하고, 함수로 표현 가능

9. 타입스크립트 만의 타입

구분	Type Alias	Interface
정의	새로운 타입을 정의할 수 있음	객체 구조를 정의할 수 있음
확장성	Intersection 및 Union 타입을 정의할 수 있음	인터페이스 확장(extends)을 통해 확장 가능
혼합	다른 타입과 혼합할 수 있음	인터페이스 간 혼합(merge)이 가능
직접 수정	기존 타입을 직접 수정할 수 없음	기존 인터페이스에 필드/메서드를 추가할 수 있음
선언 병합	불가능	가능

사용처	기본 타입 재정의, 유니온/인터섹션 타입	객체 구조 정의, API 정의, 클래스 구현

10. 호출 시그니처

함수의 입력값과 출력값의 구조를 나타내는 타입 표현입니다.

호출 시그니처는 함수의 사용 방법과 동작을 명확히 정의하여 함수의 사용성과 안정성을 높입니다.

호출 시그니처의 주요 구성 요소는 다음과 같습니다:

1. 매개변수 선언:
   - 함수에 전달되는 입력값의 타입과 이름을 정의합니다.
   - 선택적 매개변수, 나머지 매개변수 등을 지원합니다.
2. 반환 타입 선언:
   - 함수의 반환값 타입을 명시합니다.
   - 반환값이 없는 경우 void를 사용할 수 있습니다.
3. 제네릭 타입 파라미터(선택):
   - 함수가 다양한 타입의 값을 처리할 수 있도록 타입 파라미터를 정의할 수 있습니다.