주석, 변수, 상수, 타입

- 주석(comment)

```
// 내용       : 한 줄 주석
/* 내용 */    : 여러 줄로 표시할 경우 사용하는 주석으로 /* 와 */ 사이의 모든 주석
```

- 타입(type)
  1. 숫자(num)
     a. int : 정수
     b. double : 실수
  2. 문자열(String)
     - 홀따옴표(')와 쌍따옴표(")를 혼용해서 사용
  3. 논리(bool)
     - true 와 false 를 가짐
  4. Record
     - 정의 및 특징
       - 서로 다른 타입의 데이터를 모아놓은 자료형
       - 각 타입마다 이름으로 접근이 가능하며 값을 변경할 수 없는 불변성을 가짐
     - 선언
       1. positional params
          a. 값이 저장되는 위치별 타입을 지정하는 방식

```
(String, int) hong = ('홍길동', 30);

 print('이름: ${hong.$1}');
 print('나이: ${hong.$2}');
```

```
이름: 홍길동
나이: 30
```

       2. named params
          a. 값에 이름을 지정하는 방식

```
({String name, int age}) hong = (name: '홍길동', age : 30);
// 네임드 파라미터를 이용하는 경우 값의 위치와 무관
// ({String name, int age}) hong = ( age : 30, name: '홍길동');

print('이름: ${hong.name}');
print('나이: ${hong.age}');
```

```
이름: 홍길동
나이: 30
```

       3. positional params 와 names params

```
  (int, {String name, int age}) hong = (100, name: '홍길동', age: 30);

  print('번호: ${hong.$1}');
  print('이름: ${hong.name}');
  print('나이: ${hong.age}');
}
```

```
번호: 100
이름: 홍길동
나이: 30
```

- 클래스와 비교

Record	Class
불변성	선택적 불변성
Stack 영역	Heap 영역
- 데이터의 생명주기가 짧은 경우 - 불변성을 명시할 경우 - 절대로 상속하지 않는 경우	- 데이터의 생명주기가 전역 범위에 가까울 경우 - 확장성이나 상속이 필요한 경우 - 다형성이 요구되는 경우

5. List
   - Dart의 배열에 해당하는 자료형으로 [](대괄호)를 활용해서 선언

```
var myList = [1, 3, 5];
```

6. Set
   - 집합을 나타내는 자료형으로 {}(중괄호)를 활용해서 선언

```
var mySet = { 1, 3, 5 };
```

   - 값의 순서가 없고 중복이 허용되지 않음
7. Map
   - 키와 값을 한 쌍으로 가지는 자료형으로 {}(중괄호)를 활용해서 선언

```
var myMap = {
    "홍길동" : "컴퓨터공학",
    "강대한" : "경영학"
};
```

   - 값은 중복이 허용되나 키는 중복될 수 없음
8. Rune
   - 문자열을 구성하는 각 문자의 유니코드 값을 나타내는 자료형
9. Symbol
   - Dart 내에 선언된 식별자나 연산자를 표현하는 데 쓰이는 자료형
10. Null

- 변수
  - 선언 및 초기화
    - 타입 변수명 = 초기값;
  - 값
    - 데이터가 저장된 위치(reference)

> Dart의 데이터는 모두 객체로 사실상 모든 타입이 참조타입이다.

- 유연한 자료형 선언

num	- int 와 double의 supertype으로 int와 double을 모두 참조가능 ⇒ int 와 double 사이의 타입 변환 불가능
var	- 타입 미지정 - 초기화 후 타입 변경 불가능
dynamic	- 타입 미지정 - 초기화 이후에도 타입 변경 가능

- 상수

> 리터럴은 값 그자체이고 상수는 그 값을 가진 후 값이 변경되지 않는 변수

1. final
   - 런타임하는 시점에 상수로 처리되므로 필요에 따라 변수 값이나 함수를 실행하는 결과값으로 초기화가 가능
2. const
   - 컴파일하는 시점에 상수로 처리되므로 사실상 리터럴만 가질 수 잇음
     함수를 호출한 결과값으로 초기화는 불가능
