Kotlin 치트시트

변수, 타입, Null 안정성

- 변수 선언:
  - `val`: 읽기 전용(Immutable) 변수입니다. 한 번 값을 할당하면 변경할 수 없으며, Java의 `final`과 유사합니다.
  - `var`: 재할당이 가능한(Mutable) 변수입니다.
- 타입 추론: `val name = "Kotlin"`과 같이 작성하면 컴파일러가 초기값을 바탕으로 타입을 자동으로 추론합니다.
- 기본 데이터 타입: `Int`, `Double`, `Boolean`, `Char`, `String` 등을 제공합니다. Kotlin에서는 Java와 달리 모든 타입을 객체로 취급합니다.
- Null 안정성 (Null Safety): 객체 지향 언어의 고질적인 문제인 Null Pointer Exception(NPE)을 컴파일 단계에서 방지할 수 있는 Kotlin의 핵심 기능입니다.
  - `String?`: Null 값을 가질 수 있는 타입입니다.
  - `String`: Null 값을 가질 수 없는 타입입니다.
- 안전한 호출 (Safe Call): `name?.length`와 같이 사용하며, `name`이 `null`일 경우 에러 대신 `null`을 반환합니다.
- 엘비스 연산자 (Elvis Operator): `name?.length ?: -1`과 같이 사용하며, 좌항이 `null`일 경우 우항의 기본값(-1)을 반환합니다.
- Not-null 단언 (Not-null Assertion): `name!!.length`와 같이 사용하며, `null`이 아님을 강제로 보증합니다. (만약 `null`일 경우 NPE가 발생하므로 사용에 주의해야 합니다)

제어 흐름

- `if-else` 조건문: Java와 유사하지만, Kotlin에서는 구문(Statement)이 아닌 표현식(Expression)으로 사용하여 직접 값을 반환할 수 있습니다.

```
val max = if (a > b) a else b
```

- `when` 선택문: Java의 `switch` 문을 대체하며, 범위 검사, 타입 확인 등 훨씬 더 강력하고 유연한 조건 처리가 가능합니다.

```
when (x) {
    1 → print("x는 1입니다")
    2, 3 → print("x는 2 또는 3입니다")
    in 4..7 → print("x는 4에서 7 사이의 범위에 있습니다")
    is String → print("x는 String 타입입니다")
    else → print("어떤 조건에도 해당하지 않습니다")
}
```

- `for` 반복문: 범위나 컬렉션을 순회할 때 유용하게 사용됩니다.

```
for (item in collection) print(item)
for (i in 1..5) { ... } // 1부터 5까지 (포함)
for (i in 1 until 5) { ... } // 1부터 4까지 (5 제외)
for (i in 5 downTo 1 step 2) { ... } // 5부터 1까지 2씩 감소하며 순회
```

- `while` / `do-while`: Java의 반복문 구조와 동일하게 동작합니다.

함수

- 함수 정의: `fun` 키워드를 사용하여 정의하며, 매개변수와 반환 타입을 명시합니다.

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

- 단일 표현식 함수: 함수의 본문이 단일 식인 경우 `=`를 사용하여 더 간결하게 정의할 수 있습니다.
- 기본 인자 (Default Arguments): 매개변수에 기본값을 지정하여 함수 호출 시 인자를 생략할 수 있게 합니다.
- 명명된 인자 (Named Arguments): 함수 호출 시 인자의 이름을 명시하여 순서와 상관없이 인자를 전달할 수 있습니다.

클래스와 객체

- 클래스 정의: 클래스 선언과 동시에 주 생성자(Primary Constructor)를 정의할 수 있습니다.

```
class Person(val name: String) {
    var age: Int = 0
}
```

- 데이터 클래스 (Data Class): 데이터를 저장하는 목적으로 사용되며, `equals()`, `hashCode()`, `toString()`, `copy()` 메서드를 자동으로 생성해 줍니다.

```
data class User(val name: String, val age: Int)
```

- 상속: Kotlin의 클래스는 기본적으로 상속이 금지되어 있으며, 상속을 허용하려면 `open` 키워드를 명시해야 합니다.
- 인터페이스: 추상 메서드뿐만 아니라 구현이 포함된 메서드도 가질 수 있습니다.
- 객체 선언 (Object): 별도의 클래스 정의 없이 싱글턴(Singleton) 패턴의 객체를 즉시 생성할 수 있습니다.

확장 기능 (Extensions)

상속이나 데코레이터 패턴을 사용하지 않고도 기존 클래스에 새로운 함수나 프로퍼티를 멤버인 것처럼 추가할 수 있는 기능입니다.

```
fun String.initials(): String {
    // 공백으로 분리 후 각 단어의 첫 글자만 추출하여 합칩니다.
    return this.split(' ').map { it.first() }.joinToString("")
}

val name = "John Doe"
println(name.initials()) // 결과: JD
```

고차 함수와 람다

- 람다 표현식: 중괄호 내부에 `{ 인자 -> 본문 }` 형식으로 작성하는 익명 함수입니다.
- 고차 함수: 함수를 인자로 받거나 결과로 함수를 반환하는 함수를 말합니다.
- 컬렉션 처리 함수: `filter`, `map`, `reduce` 등 함수형 스타일의 풍부한 API를 제공합니다.

```
val numbers = listOf(1, 2, 3, 4, 5)
val evens = numbers.filter { it % 2 == 0 } // 짝수만 필터링: [2, 4]
val squared = numbers.map { it * it } // 각 요소를 제곱: [1, 4, 9, 16, 25]
```

코루틴 (Coroutines)

비동기 프로그래밍을 마치 동기 코드처럼 직관적이고 효율적으로 작성할 수 있도록 돕는 경량 스레드 모델입니다.

- `suspend` 함수: 현재 스레드를 차단하지 않고 실행을 일시 중단할 수 있는 비동기 함수입니다.
- 코루틴 빌더:
  - `launch`: 결과를 반환하지 않는 비동기 작업을 시작합니다.
  - `async`: 비동기 작업을 시작하고 `Deferred` 객체를 반환하며, `await()`를 통해 결과를 수신합니다.
  - `runBlocking`: 코루틴 내의 모든 작업이 완료될 때까지 현재 스레드를 대기시킵니다.

스코프 함수 (Scope Functions)

특정 객체의 컨텍스트 내에서 일시적인 스코프를 형성하여 코드를 더욱 간결하고 가독성 있게 작성하도록 돕습니다.

- `let`: 객체가 `null`이 아닐 때 특정 작업을 수행하고 결과를 반환할 때 주로 사용합니다.
- `apply`: 객체의 속성을 설정하고 설정된 객체 자신을 다시 반환할 때 유용합니다. (객체 초기화에 자주 사용)
- `run`, `with`, `also`: 상황에 따라 객체를 참조하는 방식과 반환값이 다르므로 적절히 선택하여 사용합니다.