GUÍA DEFINITIVA DE KOTLIN PARA EL EXAMEN

Resumen de conceptos, funciones y trucos (Strings, Listas, Matrices, Data Classes).

1. ORDENACIÓN Y BÚSQUEDA (Los ".sort" y amigos)

Esta es la parte más importante para sacar nota alta. Sirve para Listas y Arrays.

A. Ordenar (Sorting)

Hay dos formas de ordenar:

1. **Modificar la lista original** (acaba en nada): lista.sort()
2. **Crear una lista nueva ordenada** (acaba en "ed"): val nueva = lista.sorted()

Función	Qué hace	Ejemplo
.sort()	Ordena la lista original de menor a mayor.	numeros.sort()
.sorted()	Devuelve **otra** lista ordenada, la original no cambia.	val ord = nums.sorted()
.sortDescending()	Ordena la original de mayor a menor (Z-A).	numeros.sortDescending()
.sortedDescending()	Devuelve otra lista ordenada de mayor a menor.	val ord = nums.sortedDescending()
.sortBy { it.propiedad }	Ordena objetos según una propiedad (ej: precio).	productos.sortBy { it.precio }
.sortedBy { it.propiedad }	Devuelve lista ordenada por propiedad.	val ranking = alumnos.sortedBy { it.nota }

B. Búsqueda y Análisis (Finding)

Estas funciones devuelven un dato o un booleano, no una lista.

Función	Qué hace
.max() / .maxOrNull()	Devuelve el valor más alto (números).
.min() / .minOrNull()	Devuelve el valor más bajo.
.maxByOrNull { it.propiedad }	Devuelve **el objeto** que tiene el valor más alto en esa propiedad. Ej: alumnos.maxByOrNull { it.nota } (El alumno con mejor nota).
.minByOrNull { it.propiedad }	Devuelve el objeto con el valor más bajo.
.find { condicion }	Devuelve el **primer** elemento que cumple la condición. Si no, devuelve null.
.contains(elemento)	Devuelve true si el elemento está en la lista.
.any { condicion }	Devuelve true si **al menos uno** cumple la condición.
.all { condicion }	Devuelve true si **TODOS** cumplen la condición.
.none { condicion }	Devuelve true si **NINGUNO** cumple la condición (o si está vacía).
.count { condicion }	Cuenta cuántos elementos cumplen la condición.

C. Transformación y Filtrado (Functional Programming)

Estas funciones crean listas nuevas basadas en la original.

Función	Qué hace	Ejemplo
.filter { condicion }	Crea una lista solo con los que cumplen la condición.	aprobados = lista.filter { it.nota >= 5 }
.map { transformacion }	Convierte cada elemento	nombres = alumnos.map {

	en otra cosa.	it.nombre } (Pasa de lista de Alumnos a lista de Strings)
.reversed()	Devuelve la lista al revés.	[1, 2, 3] -> [3, 2, 1]
.distinct()	Elimina duplicados.	[1, 2, 2, 1] -> [1, 2]

2. STRINGS (Texto)

Un String es casi como una lista de caracteres (Char).

- **Acceso:** texto[0] es la primera letra.
- **Longitud:** texto.length. Último índice: texto.lastIndex (o length - 1).

Funciones Clave:

- .uppercase() / .lowercase(): Todo a mayúsculas o minúsculas.
- .trim(): Quita espacios del principio y final. Vital para limpiar inputs sucios.
- .split(" "): Corta el texto por el separador y devuelve una **Lista de Strings**.
  - Ej: "Hola Mundo".split(" ") -> ["Hola", "Mundo"]
- .replace("viejo", "nuevo"): Cambia caracteres o palabras.
- .substring(inicio, fin): Corta un trozo. **Cuidado:** El final es exclusivo (no se incluye).
- .toInt(): Convierte texto a número. Si falla da error (usa try-catch o toIntOrNull()).
- .toCharArray(): Convierte el String en un Array de caracteres modificable.

3. LISTAS (List vs MutableList)

La distinción más importante de Kotlin.

- **List (Inmutable):** listOf(...).
  - NO tiene .add(), .remove(), .clear().
  - Solo sirve para leer datos fijos.
- **MutableList (Modificable):** mutableListOf(...).
  - SÍ puedes añadir, borrar y cambiar cosas.
  - Es la que usarás en el 90% de los exámenes.

Funciones de MutableList:

- .add(elemento): Añade al final.
- .add(posicion, elemento): Inserta en medio (desplaza los demás).
- .remove(objeto): Borra la primera aparición del objeto.
- .removeAt(indice): Borra lo que hay en esa posición.
- .set(posicion, valor) o lista[pos] = valor: Sobreescribe.
- .clear(): Borra todo.

4. MATRICES (Listas de Listas)

No existe el tipo "Matriz" en Kotlin básico. Usamos MutableList<MutableList<Int>>.

Conceptos Clave:

- **Fila:** matriz[i] (Es una lista horizontal completa).
- **Celda:** matriz[i][j] (Fila i, Columna j).
- **Recorrer:** Siempre con dos bucles anidados.

```
for (i in 0 until filas) {       // Bucle i: Filas (Vertical)
   for (j in 0 until columnas) { // Bucle j: Columnas (Horizontal)
      print(matriz[i][j])
   }
   println()
}
```

- **Cuadrada:** Filas == Columnas.
- **Diagonal Principal:** Cuando i == j (0,0), (1,1)...
- **Diagonal Secundaria:** Cuando i + j == tamaño - 1.

5. DATA CLASSES (Objetos)

Para guardar datos complejos juntos (Alumno, Producto, Coche).

data class Alumno(val nombre: String, var nota: Double)

La Magia Automática:

1. **toString():** Al hacer println(alumno) sale bonito: Alumno(nombre=Juan, nota=8.0).
2. **equals() (==):** Compara el contenido. alumno1 == alumno2 es true si los datos son iguales.
3. **copy():** Crea un clon modificando algo. Vital porque val no deja cambiar.
   - val alumnoMejorado = alumno.copy(nota = 10.0)

LA TRAMPA DEL EXAMEN:
Si defines una propiedad fuera del paréntesis (dentro de { ... }):

- NO sale en el toString().
- NO se compara en el equals().
- **NO SE COPIA** con el copy() (se reinicia al valor por defecto).
- *Consejo:* Mete todo en el paréntesis (...) a menos que te pidan explícitamente lo contrario.

6. LECTURA (Scanner)

El patrón estándar para leer en Kotlin en exámenes.

```
val scan = Scanner(System.`in`).useLocale(Locale.US) // Para leer decimales con punto (9.5)

val entero = scan.nextInt()
val decimal = scan.nextDouble()
val palabra = scan.next()      // Lee hasta el primer espacio
val frase = scan.nextLine()    // Lee toda la línea (CUIDADO con el buffer)
```

El problema del Buffer (nextLine):
Si haces nextInt() y luego nextLine(), el nextLine se "come" el Enter que sobró del número y parece que se salta la lectura.

- **Solución:** Pon un scan.nextLine() extra "fantasma" después de leer números si luego vas a leer frases.