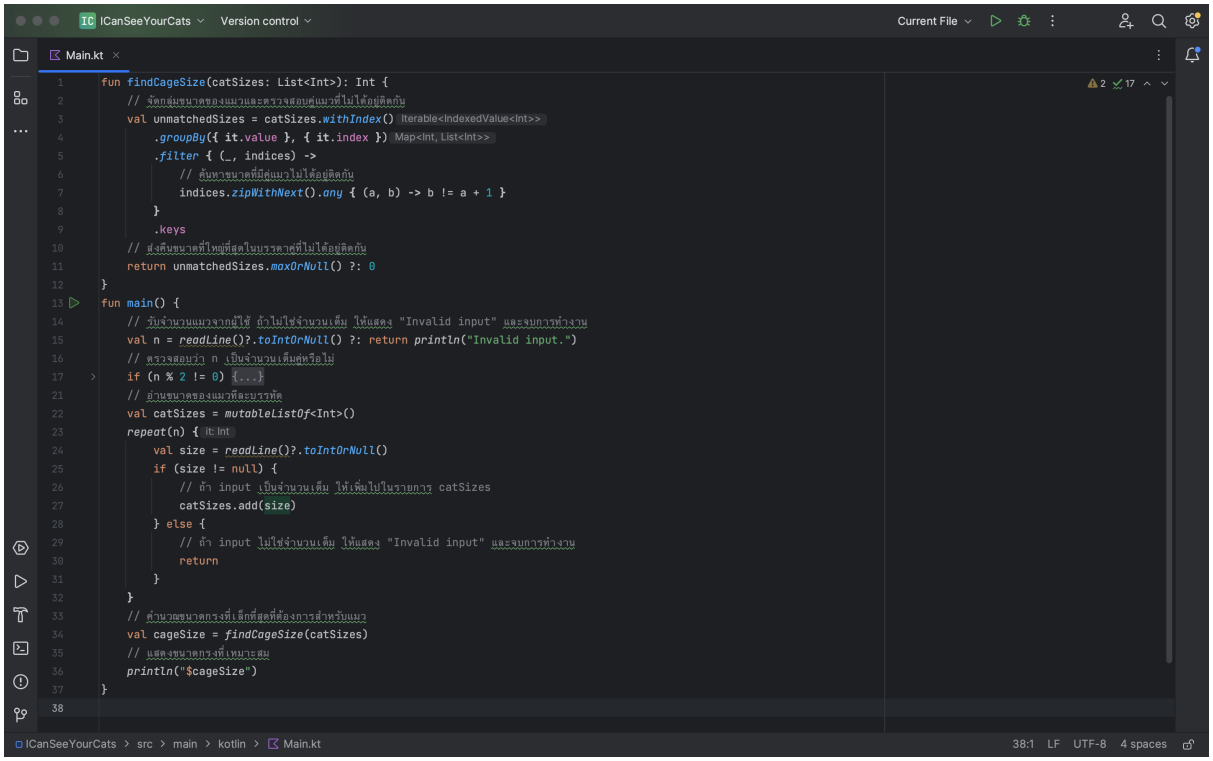

โค้ดนี้ใช้ **Functional Programming Paradigm** ในการแก้ไขปัญหา โดย Kotlin รองรับแนวคิดแบบฟังก์ชันและเป็นภาษาที่สามารถใช้เขียนเชิงฟังก์ชันได้ดี การเลือกใช้ฟังก์ชันเพื่อลดความซับซ้อนและทำให้โค้ดเข้าใจได้ง่ายขึ้นเป็นแนวคิดที่ดีในบริบทนี้

Paradigm ที่ใช้: Functional Programming

- **Declarative Style**: โค้ดในฟังก์ชัน findCageSize ใช้ฟังก์ชันสำเร็จรูปของ Kotlin เพื่อทำงานกับข้อมูล ทำให้เราไม่จำเป็นต้องระบุขั้นตอนละเอียด ลดการใช้คำสั่งแบบ procedural ซึ่งสอดคล้องกับการเขียนโปรแกรมเชิงฟังก์ชัน
- **Stateless Computations**: ฟังก์ชัน findCageSize ไม่ได้เปลี่ยนแปลงข้อมูลใน catSizes แต่จะรับข้อมูลมาและคืนค่าผลลัพธ์โดยไม่แก้ไขข้อมูลเดิม ซึ่งช่วยให้โค้ดอ่านง่ายขึ้นและลดปัญหาข้อผิดพลาดที่อาจเกิดขึ้นจากการเปลี่ยนแปลงข้อมูลภายนอก
- **Higher-Order Functions**: ฟังก์ชันที่ใช้ เช่น groupBy, filter, zipWithNext, และ maxOrNull เป็น higher-order functions ที่ใช้ในการจัดการข้อมูลในลักษณะของ functional programming ซึ่งช่วยให้โค้ดดูสะอาดและสั้นลง ไม่ต้องใช้การวนลูปเพื่อคำนวณแบบเดิม

แนวทางการแก้ปัญหาในโค้ด

โค้ดนี้มีขั้นตอนการแก้ปัญหาที่แบ่งเป็นลำดับ ดังนี้:

1. **รับ Input**: รับจำนวนแมว n จากผู้ใช้ ซึ่งจะต้องเป็นจำนวนเต็มคู่ หากไม่ใช่จำนวนเต็มคู่จะให้ผู้ใช้ป้อนใหม่ จากนั้นรับขนาดของแมวทีละตัว (แต่ละบรรทัด) ไปเก็บใน List catSizes
2. **การประมวลผล** (findCageSize):
    - **Grouping**: ใช้ฟังก์ชัน groupBy เพื่อจัดกลุ่มแมวตามขนาด เพื่อที่จะระบุคู่ของแมวที่มีขนาดเท่ากัน
    - **Filtering Non-adjacent Pairs**: ใช้ zipWithNext เพื่อตรวจสอบว่ามีคู่แมวที่ไม่ได้อยู่ติดกันหรือไม่ หากพบว่ามีคู่แมวที่ไม่ติดกัน จะเก็บขนาดของแมวคู่นั้นไว้ใน unmatchedSizes
    - **Find Maximum Size**: ใช้ maxOrNull เพื่อตรวจสอบขนาดที่ใหญ่ที่สุดในบรรดาคู่แมวที่ไม่ได้อยู่ติดกัน หากไม่มีคู่ที่ต้องเคลื่อนย้าย ผลลัพธ์จะเป็น 0
3. **แสดงผลลัพธ์**: ขนาดของกรงที่เหมาะสมที่สุดจะถูกพิมพ์ออกมาในรูปแบบขนาดของกรงที่เล็กที่สุดที่สามารถเคลื่อนย้ายแมวได้ทั้งหมด โดยที่ทำให้คู่แมวอยู่ติดกัน