📚 Module 19: Coordinate System - Complete Notes

🎯 What You'll Learn

Master the **Cartesian coordinate system** — representing positions with (x, y) pairs. Essential for UI testing, grid operations, and spatial data.

📖 Concept Explained

The Basics

A coordinate system uses two axes to define position:

- **X-axis** → Horizontal (left/right)
- **Y-axis** → Vertical (up/down)
- **Origin** → (0, 0) where axes meet

```
              (+y)
                |
       Q2       |        Q1
     (-x,+y)    |      (+x,+y)
                |
  ──────────────┼─────────────→ (+x)
                |
       Q3       |        Q4
     (-x,-y)    |      (+x,-y)
                |
              (-y)
```

💻 Programming Connection

Code Examples

```
# Example 1: Point Representation

# As tuple
point1 = (3, 5)
x, y = point1

# As class
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"
```

```python
p = Point(3, 5)
print(p.x, p.y)  # 3 5
```

```python
# Example 2: Quadrant Detection

def get_quadrant(x, y):
    """Determine which quadrant a point is in"""
    if x == 0 or y == 0:
        return "on axis"
    elif x > 0 and y > 0:
        return "Q1"
    elif x < 0 and y > 0:
        return "Q2"
    elif x < 0 and y < 0:
        return "Q3"
    else:
        return "Q4"

print(get_quadrant(3, 5))   # Q1
print(get_quadrant(-2, 3))  # Q2
```

```python
# Example 3: Grid Operations

def create_grid(rows, cols, default=None):
    """Create 2D grid"""
    return [[default for _ in range(cols)] for _ in range(rows)]

def get_cell(grid, row, col):
    return grid[row][col]

def set_cell(grid, row, col, value):
    grid[row][col] = value

# 3x4 grid
grid = create_grid(3, 4, 0)
set_cell(grid, 1, 2, 'X')
print(grid)
```

```python
# Example 4: Movement/Navigation

def move(position, direction, steps=1):
    """Move from position in direction"""
    x, y = position
    directions = {
        'up': (0, 1), 'down': (0, -1),
        'left': (-1, 0), 'right': (1, 0)
    }
    dx, dy = directions[direction]
    return (x + dx * steps, y + dy * steps)
```

```
pos = (0, 0)
pos = move(pos, 'right', 3)  # (3, 0)
pos = move(pos, 'up', 2)     # (3, 2)
```

🧪 SDET/Testing Application

```python
# SDET Scenario: UI Element Position Testing

def is_within_viewport(element_pos, element_size, viewport_size):
    """Check if element is fully visible in viewport"""
    x, y = element_pos
    w, h = element_size
    vw, vh = viewport_size

    return {
        "visible": 0 <= x and x + w <= vw and 0 <= y and y + h <= vh,
        "element_bounds": (x, y, x + w, y + h),
        "viewport": (0, 0, vw, vh)
    }

result = is_within_viewport((100, 50), (200, 100), (1920, 1080))
print(f"Visible: {result['visible']}")  # True
```

🔑 Key Takeaways

✅ **(x, y)** — Unique position identifier
✅ **Grid[row][col]** — 2D array access pattern
✅ **Movement** — Add direction vectors

💾 *Save as:* `Module_19_Coordinate_System.md`