# CH1

1. **หน้า 1-2: บทนำและสารบัญ**
    - อธิบายเกี่ยวกับบทนำและหัวข้อทั้งหมดที่จะกล่าวถึงในเอกสารนี้ เช่น การแก้ปัญหาในชีวิตประจำวัน, แนวคิดในการแก้ปัญหา, ประเภทของปัญหา, การใช้คอมพิวเตอร์ช่วยแก้ปัญหา, ปัญหาที่พบในการแก้ปัญหา, และการวัดผลและทดสอบ.
2. **หน้า 3-4: การแก้ปัญหาในชีวิตประจำวัน**
    - อธิบายปัญหาทั่วไปที่เราพบในชีวิตประจำวัน ตั้งแต่ปัญหาง่ายๆ เช่น เลือกอาหารที่จะกิน ไปจนถึงปัญหาที่ยากกว่า เช่น การเลือกอาชีพหลังจบการศึกษา.
3. **หน้า 5-7: คำถามในการแก้ปัญหา (5W1H)**
    - นำเสนอการใช้คำถาม 5W1H (ใคร, อะไร, ที่ไหน, เมื่อไหร่, ทำไม, อย่างไร) เพื่อช่วยในการวิเคราะห์และแก้ปัญหาอย่างเป็นระบบ.

    What (อะไร): ต้องระบุปัญหาอย่างชัดเจนว่าเป็นอะไร

    Who (ใคร): ต้องพิจารณาว่าใครเกี่ยวข้องหรือได้รับผลกระทบ

    When (เมื่อไร): ช่วงเวลาที่ปัญหาเกิดขึ้น

    Where (ที่ไหน): สถานที่ที่ปัญหาเกิดขึ้น

    Why (ทำไม): สาเหตุที่ทำให้ปัญหาเกิดขึ้น

    How (อย่างไร): วิธีการที่จะใช้ในการแก้ไขปัญหานั้น

นี่คือตัวอย่างคำถาม 5W1H สำหรับการแก้ปัญหาในสองเรื่องที่แตกต่างกัน:

**1. เรื่องการเปิดตัวผลิตภัณฑ์ใหม่**

**Who (ใคร)**: ใครคือเป้าหมายหลักสำหรับผลิตภัณฑ์ใหม่นี้?

**What (อะไร)**: ผลิตภัณฑ์ใหม่นี้แก้ปัญหาอะไรให้กับลูกค้า?

**Where (ที่ไหน)**: ที่ไหนจะเป็นสถานที่จำหน่ายหลักสำหรับผลิตภัณฑ์นี้?

**When (เมื่อไหร่)**: เมื่อไหร่ที่เหมาะสมที่สุดสำหรับการเปิดตัวผลิตภัณฑ์?

**Why (ทำไม)**: ทำไมต้องพัฒนาผลิตภัณฑ์นี้? มีปัจจัยอะไรที่สนับสนุนการสร้างมันขึ้นมา?

**How (อย่างไร)**: จะใช้วิธีการใดในการโปรโมตและจำหน่ายผลิตภัณฑ์นี้ให้กับตลาด?

**2. เรื่องการปรับปรุงระบบการให้บริการลูกค้า**

**Who (ใคร)**: ใครคือผู้ที่จะรับผิดชอบในการปรับปรุงระบบนี้?

**What (อะไร)**: ปัญหาหลักที่ลูกค้าพบกับระบบปัจจุบันคืออะไร?

**Where (ที่ไหน)**: จะปรับปรุงส่วนใดบ้างในระบบการให้บริการ?

**When (เมื่อไหร่)**: เมื่อไหร่ที่ควรเริ่มต้นโปรเจคปรับปรุงระบบนี้?

**Why (ทำไม)**: ทำไมเราถึงต้องปรับปรุงระบบการให้บริการนี้? มีผลตอบรับอย่างไรจากลูกค้า?

**How (อย่างไร)**: จะใช้วิธีการและเทคโนโลยีใดในการปรับปรุงระบบการให้บริการนี้?

4. **หน้า 8-10: การคิด**

- กล่าวถึงวิธีการคิดที่แตกต่างกัน รวมถึงการคิดแบบดั้งเดิมและการคิดนอกกรอบ พร้อมด้วยตัวอย่างเฉพาะเช่น การแก้ปัญหาการเข้าสู่ระบบ.

**หน้า 8**

หน้านี้มุ่งเน้นไปที่การแนะนำแนวทางการคิดเพื่อแก้ปัญหา:

- **การคิดเป็นขั้นตอน**: การใช้ข้อมูลที่มีอยู่เพื่อแก้ปัญหาหรือหาคำตอบ นี่คือการคิดที่ต้องการความชัดเจนในทิศทางและแนวทางในการแก้ไข ตัวอย่างเช่น การตรวจสอบชื่อผู้ใช้และรหัสผ่านเมื่อมีปัญหาในการเข้าสู่ระบบ หรือการตรวจสอบการเชื่อมต่ออินเทอร์เน็ต หรือการรีเซ็ตรหัสผ่าน เป็นการแก้ไขที่เป็นลำดับขั้นตอนและตามกระบวนการที่แน่นอน.
- **การคิดนอกกรอบ**: แนวทางการคิดที่ไม่จำกัดอยู่เพียงวิธีการแบบเดิมๆ แต่ใช้วิธีการที่สร้างสรรค์และไม่ตามแบบแผน เช่น เมื่อเจอปัญหาในการเข้าสู่ระบบอาจไม่จำเป็นต้องตรวจสอบเพียงชื่อ

ผู้ใช้หรือรหัสผ่าน แต่อาจคิดถึงการเปลี่ยนแปลงวิธีการล็อกอิน เช่น ใช้การยืนยันตัวตนด้วยใบหน้าหรือการพิสูจน์ตัวตนจากอีเมล.

**หน้า 9**

- **ต่อยอดจากหน้า 8** หน้านี้เน้นเทคนิคการคิดและการประยุกต์ใช้ในการแก้ปัญหา:
    - การใช้ข้อมูลและทรัพยากรที่มีอยู่เพื่อหาวิธีแก้ปัญหาในแบบที่เป็นระบบและมีลำดับขั้นตอน.
    - การสำรวจวิธีการแก้ไขที่อาจไม่ได้มาจากกระบวนการปกติ แต่อาจผ่านการทดลองและข้อผิดพลาดหรือการใช้แนวคิดสร้างสรรค์ใหม่ๆ ที่อาจแตกต่างจากแนวทางเดิม.

  หน้า 8-9 ในสารบัญนี้จึงเป็นการทำความเข้าใจถึงการใช้ทักษะการคิดที่หลากหลายในการแก้ปัญหา โดยทั้งหน้ามุ่งเน้นไปที่การพัฒนาความสามารถในการใช้ข้อมูลที่มี การเปิดรับต่อการคิดอย่างเสรี และการสร้างสรรค์แนวทางใหม่ๆ ในการจัดการกับปัญหาต่างๆ ที่เกิดขึ้น.

**หน้า 10**

**ทักษะการแก้ปัญหา (Problem Solving Skills)**

**หน้านี้มุ่งเน้นไปที่การนิยามและการอธิบายทักษะที่จำเป็นในการแก้ปัญหา:**

- การกำหนดทักษะ: ทักษะการแก้ปัญหาถูกกำหนดว่าเป็นทักษะที่เกี่ยวข้องกับการหาวิธีการแก้ไขปัญหาโดยการพิจารณารายละเอียดทั้งหมดของปัญหา, การวิเคราะห์ปัญหานั้น, และการค้นหาทรัพยากรที่จำเป็นเพื่อเสร็จสิ้นงาน.
- การพัฒนาทักษะ: บ่งชี้ว่าการแก้ปัญหาที่มีประสิทธิภาพต้องอาศัยการมองเห็นรายละเอียดที่ครบถ้วน, การวิเคราะห์อย่างถ่องแท้, และการหาทรัพยากรที่เหมาะสมเพื่อตอบสนองต่อปัญหานั้น.

**ความสำคัญของทักษะการแก้ปัญหา**

- การมีทักษะการแก้ปัญหาที่ดีจะช่วยให้บุคคลสามารถรับมือกับสถานการณ์ที่ต้องการการตัดสินใจหรือการหาคำตอบได้ดีขึ้น ไม่ว่าจะเป็นในชีวิตประจำวันหรือในสถานการณ์ทางธุรกิจและอาชีพ.

- การเสริมสร้างทักษะเหล่านี้จะทำให้บุคคลมีความสามารถในการแก้ไขปัญหาที่ซับซ้อนมากขึ้นและสามารถนำไปใช้ในหลากหลายสถานการณ์ได้อย่างมีประสิทธิภาพ.

ในหน้านี้, ทักษะการแก้ปัญหาถูกนำเสนอเป็นทักษะพื้นฐานที่จำเป็นสำหรับทุกคนในยุคปัจจุบัน ที่ต้องเผชิญกับปัญหาและความท้าทายที่หลากหลาย และต้องมีการตัดสินใจอย่างมีข้อมูลครบถ้วน.

5. **หน้า 11-18: แนวคิดทั่วไปในการแก้ปัญหา**
    - ขั้นตอนต่างๆ ในกระบวนการแก้ปัญหาตั้งแต่การระบุปัญหาไปจนถึงการประเมินผลของวิธีการแก้ปัญหาที่เลือกไว้.

**ขั้นตอนที่ 1: ระบุปัญหา (Identify the Problem)**

- **การระบุปัญหา**: ขั้นตอนนี้เน้นที่การตั้งชื่อและกำหนดปัญหาอย่างชัดเจน ซึ่งจำเป็นต้องทำก่อนที่จะเริ่มแก้ไขปัญหาใดๆ มันเกี่ยวข้องกับการระบุว่าปัญหานั้นคืออะไร และการพิจารณาถึงผลกระทบหรือสิ่งที่ต้องการแก้ไข.

**ขั้นตอนที่ 2: เข้าใจปัญหา (Understand the Problem)**

- **การเข้าใจปัญหา**: ขั้นตอนนี้คือการทำความเข้าใจปัญหาอย่างลึกซึ้ง รวมถึงสาเหตุและปัจจัยที่เกี่ยวข้องกับปัญหา เป้าหมายคือการทำให้ชัดเจนว่าปัญหานั้นประกอบด้วยอะไรบ้างและมีบริบทหรือขอบเขตอย่างไร.

**ขั้นตอนที่ 3: ระบุทางเลือกในการแก้ปัญหา (Identify Alternative Solutions)**

- **การหาทางเลือก**: ในขั้นตอนนี้จะเกี่ยวข้องกับการสร้างไอเดียหรือทางเลือกต่างๆ ที่สามารถใช้แก้ไขปัญหานั้นได้ มันอาจรวมถึงการสร้างสรรค์วิธีแก้ไขแบบต่างๆ และการประเมินว่าแต่ละทางเลือกมีความเป็นไปได้อย่างไร.

**ขั้นตอนที่ 4: เลือกทางเลือกที่ดีที่สุด (Select the Best Solution)**

- **การเลือกวิธีการแก้ไข**: ขั้นตอนนี้เน้นที่การประเมินและเลือกทางเลือกที่ดีที่สุดจากทางเลือกที่ได้มาในขั้นตอนที่แล้ว การตัดสินใจนี้จะขึ้นอยู่กับประโยชน์และข้อจำกัดของแต่ละทางเลือก รวมถึงการพิจารณาถึงความเป็นไปได้และผลกระทบในการนำไปใช้.

**ขั้นตอนที่ 5: วางแผนการดำเนินการ (Plan the Implementation)**

- **การวางแผนการดำเนินการ**: หลังจากที่เลือกทางเลือกที่ดีที่สุดแล้ว ขั้นตอนต่อไปคือการวางแผนว่าจะนำทางเลือกนั้นไปใช้งานอย่างไร มันรวมถึงการวางขั้นตอนการดำเนินการ, การจัดหาทรัพยากร, และการกำหนดเวลาสำหรับการดำเนินการ.

**ขั้นตอนที่ 6: ประเมินผลของวิธีการแก้ปัญหา (Evaluate the Solution)**

- **การประเมินผล**: ขั้นตอนสุดท้ายคือการประเมินผลลัพธ์ของการแก้ไขปัญหา เพื่อดูว่าวิธีการที่เลือกไว้นั้นได้ผลลัพธ์ที่ต้องการหรือไม่ และเพื่อตรวจสอบว่ามีข้อผิดพลาดหรือปัญหาใหม่เกิดขึ้นหรือไม่.

กระบวนการนี้ช่วยให้ผู้ที่มีปัญหาสามารถจัดการและแก้ไขปัญหาได้อย่างมีระบบและมีประสิทธิภาพ โดยแต่ละขั้นตอนมีบทบาทสำคัญในการนำไปสู่การแก้ไขปัญหาที่เป็นรูปธรรม.

6. **หน้า 19-25: การแก้ปัญหาด้วยคอมพิวเตอร์**
    - วิธีการใช้คอมพิวเตอร์ในการแก้ปัญหาและการประยุกต์ใช้ปัญญาประดิษฐ์และระบบอัลกอริธึมในการแก้ปัญหาต่างๆ.

ตัวอย่างการใช้กระบวนการแก้ปัญหา 6 ขั้นตอนในการพัฒนาโปรแกรมคำนวณ Basal Metabolic Rate (BMR) ซึ่งเป็นการวัดอัตราการเผาผลาญพลังงานขั้นต่ำที่ร่างกายต้องการเพื่อดำรงชีวิตในสภาวะพักผ่อน:

**ขั้นตอนที่ 1: ระบุปัญหา (Identify the Problem)**

- **ปัญหา**: ต้องการพัฒนาโปรแกรมคำนวณ BMR ที่ช่วยให้ผู้ใช้สามารถเข้าใจความต้องการพลังงานพื้นฐานของตนเอง เพื่อการวางแผนอาหารและการออกกำลังกายที่เหมาะสม.

**ขั้นตอนที่ 2: เข้าใจปัญหา (Understand the Problem)**

- **เข้าใจ**: การคำนวณ BMR ต้องพิจารณาหลายปัจจัย เช่น เพศ, อายุ, น้ำหนัก, ส่วนสูง และระดับการเผาผลาญพลังงาน.

**ขั้นตอนที่ 3: หาทางเลือกในการแก้ปัญหา (Identify Alternative Solutions)**

- **ทางเลือก A**: ใช้สูตร Harris-Benedict แบบเก่า

- **ทางเลือก B**: ใช้สูตร Harris-Benedict ที่ปรับปรุงใหม่
- **ทางเลือก C**: ใช้สูตร Mifflin-St Jeor ซึ่งเป็นสูตรที่ทันสมัยและแม่นยำกว่า

**ขั้นตอนที่ 4: เลือกทางเลือกที่ดีที่สุด (Select the Best Alternative Solutions)**

- **เลือกทางเลือก C**: สูตร Mifflin-St Jeor มีความแม่นยำสูงและเป็นที่ยอมรับในวงการวิทยาศาสตร์การแพทย์

**ขั้นตอนที่ 5: วางแผนการดำเนินการ (List Instructions)**

- ออกแบบ UI ให้ผู้ใช้ป้อนเพศ, อายุ, น้ำหนัก (กิโลกรัม), ส่วนสูง (เซนติเมตร)
- พัฒนาฟังก์ชันการคำนวณตามสูตร Mifflin-St Jeor:
    - สำหรับผู้ชาย: BMR = 10 * weight(kg) + 6.25 * height(cm) - 5 * age(y) + 5
    - สำหรับผู้หญิง: BMR = 10 * weight(kg) + 6.25 * height(cm) - 5 * age(y) - 161
- ทดสอบฟังก์ชันการคำนวณ

**ขั้นตอนที่ 6: ประเมินผล (Evaluate the Solution)**

- ทดสอบโปรแกรมกับข้อมูลจริงจากผู้ใช้หลายกลุ่ม
- รวบรวมข้อเสนอแนะและประเมินความแม่นยำของการคำนวณ
- ปรับปรุงฟังก์ชันการคำนวณหากจำเป็น

7. **หน้า 26-29: ประเภทของปัญหา**
    - คำอธิบายเกี่ยวกับปัญหาแบบอัลกอริธึมและฮิวริสติก และวิธีการที่ต่างกันในการแก้ปัญหาเหล่านั้น.

**หน้า 26-27: ประเภทของปัญหา (Types of Problems)**

- การนำเสนอประเภทของปัญหาต่างๆ ที่มักพบในการแก้ปัญหาด้วยคอมพิวเตอร์และในชีวิตประจำวัน สำคัญที่สุดคือการแบ่งปัญหาออกเป็นสองประเภทหลัก: อัลกอริธึมและฮิวริสติก.

- **เพิ่มเติม**: ควรทราบถึงตัวอย่างของแต่ละประเภทปัญหาเพื่อช่วยให้แยกแยะและจำแนกประเภทได้ง่ายขึ้นในการปฏิบัติจริง.

**หน้า 28: การแก้ปัญหาแบบฮิวริสติก (Heuristic solutions)**

- **คำอธิบาย**: การแก้ปัญหาแบบฮิวริสติกนี้เน้นการใช้การตัดสินใจที่อิงตามประสบการณ์และการคาดการณ์ผลลัพธ์จากข้อมูลที่มีอยู่ ไม่มีขั้นตอนที่ชัดเจนหรือสูตรเฉพาะในการแก้ไขปัญหา ซึ่งต่างจากวิธีอัลกอริทึมที่มีขั้นตอนแน่นอนและผลลัพธ์ที่คาดการณ์ได้.
- **การประยุกต์ใช้**: วิธีนี้มักใช้ในการตัดสินใจที่ซับซ้อน ซึ่งคำนึงถึงปัจจัยหลายอย่างและอาศัยการประเมินสถานการณ์จริง เช่น ในการพัฒนาแผนกลยุทธ์ธุรกิจหรือการตัดสินใจด้านการลงทุนที่ต้องพิจารณาถึงปัจจัยหลายๆ อย่างในตลาด.

    **ข้อมูลเพิ่มเติมที่ควรรู้**: การใช้วิธีการฮิวริสติกมักต้องอาศัยประสบการณ์และความรู้เฉพาะด้านของผู้ตัดสินใจ เพื่อให้สามารถวิเคราะห์และประเมินผลลัพธ์ที่อาจเกิดขึ้นได้อย่างแม่นยำยิ่งขึ้น. นอกจากนี้ การฝึกฝนและการเรียนรู้อย่างต่อเนื่องจะช่วยให้การใช้วิธีนี้มีประสิทธิภาพขึ้น.

**หน้า 29 "Algorithmic solutions" หรือการแก้ปัญหาแบบอัลกอริธึม**

ซึ่งเป็นวิธีการที่มีขั้นตอนการดำเนินงานที่ชัดเจนและเป็นระเบียบษณะของ Algorithmic solutions**:

- การแก้ปัญหาแบบอัลกอริธึมมีการปฏิบัติตามขั้นตอนที่แน่นอน จากจุดเริ่มต้นไปจนถึงจุดสิ้นสุด.
- ขั้นตอนเหล่านี้มักจะนำไปสู่ผลลัพธ์ที่ถูกต้องและมีข้อผิดพลาดน้อยมาก.
- วิธีนี้ถูกใช้ในการแก้ปัญหาทางคอมพิวเตอร์ในหลายสถานการณ์ เช่น การแก้โจทย์คณิตศาสตร์หรือการพัฒนาซอฟต์แวร์.

**ข้อมูลเพิ่มเติมที่ควรรู้**:

1. **การเข้าใจ Algorithm**: การศึกษาและเข้าใจอัลกอริธึมคือการทำความเข้าใจกระบวนการแก้ปัญหาขั้นตอนต่างๆ อย่างละเอียด รวมถึงการทำความเข้าใจว่าแต่ละขั้นตอนมีบทบาทอย่างไรในการไปถึงผลลัพธ์สุดท้าย.

2. **การออกแบบ Algorithm**: การออกแบบอัลกอริธึมที่มีประสิทธิภาพต้องพิจารณาถึงทั้งความถูกต้องและประสิทธิภาพในการดำเนินการ และต้องคำนึงถึงสถานการณ์ที่อัลกอริธึมนั้นจะถูกใช้งาน.
3. **การทดสอบและการตรวจสอบ Algorithm**: หลังจากการพัฒนาอัลกอริธึม จำเป็นต้องมีการทดสอบเพื่อตรวจสอบว่าอัลกอริธึมทำงานได้ตามที่คาดหวังและแก้ไขปัญหาที่ได้รับมอบหมายอย่างถูกต้องหรือไม่.

## 8. หน้า 30-38 Problem Solving with Computers

**หน้า 30: Problem Solving with Computers**

- หัวข้อหลัก: การใช้คอมพิวเตอร์ช่วยแก้ปัญหา
- แนวคิด: คอมพิวเตอร์เป็นเครื่องมือสำคัญที่ช่วยจัดการกับปัญหาที่ซับซ้อนและต้องการการวิเคราะห์เชิงลึก โดยเน้นการใช้กระบวนการที่เป็นระบบในการแก้ปัญหา.

---

**หน้า 31: Heuristic Solutions**

- Heuristic solutions: การแก้ปัญหาที่ใช้เหตุผลและผลลัพธ์จากความรู้และประสบการณ์.
    - ตัวอย่างการใช้งานในคอมพิวเตอร์:
        - ปัญญาประดิษฐ์ (Artificial Intelligence - AI): ใช้แก้ปัญหาที่ไม่มีคำตอบที่ชัดเจน เช่น การสร้าง chatbot ที่สามารถพูดคุยเหมือนมนุษย์ หรือการตัดสินใจในสถานการณ์ที่มีข้อมูลจำกัด.
        - AI สามารถช่วยพัฒนาความสามารถในการวิเคราะห์และตอบสนองต่อปัญหาได้เหมือนมนุษย์.
        - การใช้งาน Robotics: หุ่นยนต์ที่ใช้ในงานต่างๆ เช่น การผลิตอุตสาหกรรม หรือการช่วยงานในสถานการณ์เฉพาะ เช่น การสำรวจพื้นที่อันตราย.

เพิ่มเติมที่ควรรู้:

1. Heuristic solutions เหมาะกับปัญหาที่ไม่มีคำตอบตายตัวหรือข้อมูลไม่ครบถ้วน เช่น การวางแผนกลยุทธ์หรือการคาดการณ์.
2. การเรียนรู้วิธีนี้ช่วยพัฒนาทักษะการคิดวิเคราะห์เชิงลึกและสร้างสรรค์.

---

**หน้า 32: Algorithmic Solutions**

- Algorithmic solutions: วิธีการแก้ปัญหาที่เป็นลำดับขั้นตอนชัดเจน มีโครงสร้างที่แน่นอน.
    - รายละเอียด:
        - ใช้กระบวนการแก้ปัญหา 6 ขั้นตอน (Identify ⟶ Understand ⟶ Explore alternatives ⟶ Select ⟶ List instructions ⟶ Evaluate).
        - มีการนำขั้นตอนมาประยุกต์ในงานต่างๆ เช่น การแก้ปัญหาทางคณิตศาสตร์ หรือการพัฒนาโปรแกรมคอมพิวเตอร์.
        - อัลกอริธึมส่วนใหญ่สามารถเขียนเป็น Flowchart เพื่อแสดงขั้นตอนการทำงานได้ง่าย.

เพิ่มเติมที่ควรรู้:

1. การเขียน Algorithm ต้องมีการวางแผนล่วงหน้าและกำหนดเงื่อนไขที่ชัดเจน.
2. ตัวอย่างการใช้ Algorithm เช่น การคำนวณภาษี การจัดการระบบข้อมูล หรือการพัฒนาโปรแกรมที่ใช้การวิเคราะห์เชิงตัวเลข.

---

**หน้า 33 Problem Solving with Computers**

กระบวนการแก้ปัญหาด้วยคอมพิวเตอร์สามารถแบ่งออกเป็น 6 ขั้นตอนหลัก ดังนี้:

1. **Identify the Problem** (ระบุปัญหา): กำหนดและทำความเข้าใจปัญหาที่ต้องการแก้ไขอย่างชัดเจน.

2. **Understand the Problem** (เข้าใจปัญหา): ทำการวิเคราะห์ว่าอะไรคือข้อมูลที่เกี่ยวข้อง.
3. **Identify Alternative Ways** (ระบุทางเลือกในการแก้ปัญหา): ค้นหาวิธีแก้ปัญหาที่เป็นไปได้หลายๆ ทาง.
4. **Select the Best Solution** (เลือกวิธีที่ดีที่สุด): ประเมินข้อดี-ข้อเสีย และเลือกวิธีที่เหมาะสม.
5. **List Instructions** (เขียนขั้นตอนการแก้ปัญหา): วางลำดับขั้นตอนอย่างเป็นระบบ.
6. **Evaluate** (ประเมินผล): ตรวจสอบและวิเคราะห์ผลลัพธ์จากวิธีการแก้ปัญหาที่เลือกใช้.

---

**หน้า 34: Identify และ Understand the Problem**

**(1) Identify the Problem**

- โจทย์ปัญหาคือ การเขียนโปรแกรมเครื่องคิดเลขที่สามารถ:
    - คำนวณพื้นฐาน เช่น บวก (+), ลบ (-), คูณ (*), หาร (/), และหารเอาเศษ (%).
    - รับตัวเลข 2 จำนวน และตัวดำเนินการ.
    - ผลลัพธ์ต้องแม่นยำและรองรับข้อผิดพลาด.

**(2) Understand the Problem**

- วิเคราะห์:
    - **Input**: ตัวเลข 2 ตัว และตัวดำเนินการ (เช่น +, -, *, /, %).
    - **Output**: ผลลัพธ์ที่คำนวณได้ หรือแจ้งเตือนข้อผิดพลาด (เช่น หารด้วยศูนย์).
    - **ข้อกำหนด**: ต้องรองรับเฉพาะตัวดำเนินการที่กำหนดไว้ และต้องจัดการข้อผิดพลาด.

---

**หน้า 35 : Identify Alternative Ways และ Select the Best Solution**

**(3) Identify Alternative Ways**

- ทางเลือกในการพัฒนาโปรแกรม:
    1. ใช้ **if-else statements**: ตรวจสอบตัวดำเนินการทีละเงื่อนไข.
    2. ใช้ **dictionary mapping**: จับคู่ตัวดำเนินการกับฟังก์ชัน.
    3. ใช้ **switch-case**: สำหรับภาษาที่รองรับ (C++, Java).

**(4) Select the Best Solution**

- เลือกใช้ **if-else statements**:
    - เหมาะสำหรับผู้เริ่มต้น.
    - เข้าใจง่าย และใช้งานได้ในภาษา Python.

---

**หน้า 36 List Instructions**

**(5) List Instructions**

- เขียน Algorithm:
    1. รับ Input 2 ตัวเลขและตัวดำเนินการ.
    2. ตรวจสอบตัวดำเนินการ:
        - บวก ⟶ คำนวณผลลัพธ์การบวก.
        - ลบ ⟶ คำนวณผลลัพธ์การลบ.
        - คูณ ⟶ คำนวณผลลัพธ์การคูณ.
        - หาร ⟶ ตรวจสอบตัวเลขที่สอง (หารด้วยศูนย์ไม่ได้).
        - หารเอาเศษ ⟶ คำนวณผลลัพธ์การหารเศษ.
    3. แสดงผลลัพธ์.
    4. แจ้งข้อผิดพลาด (หากอินพุตไม่ถูกต้อง).

- สร้าง Flowchart:
    - แสดงลำดับการตรวจสอบ Input, การคำนวณ และการแสดงผล.
- เขียนโค้ดโปรแกรม:
    - ใช้ Python หรือภาษาอื่นที่ถนัด.

---

**หน้า 37 : Evaluate**

**(6) Evaluate**

- **การทดสอบโปรแกรม**:
    - **กรณีปกติ**:
        - เช่น 10 + 5 ⟶ 15, 10 / 5 ⟶ 2.
    - **กรณีพิเศษ**:
        - เช่น 10 / 0 ⟶ Error: Division by zero.
        - เช่น 10 # 5 ⟶ Error: Invalid operator.
- **การตรวจสอบผลลัพธ์ (Result Validation)**:
    - เปรียบเทียบผลลัพธ์กับเครื่องคิดเลขจริง หรือคำนวณด้วยตนเอง.

---

## 9. หน้า 39-40 Difficulties with Problem Solving

**หัวข้อหลัก: ความยากลำบากที่อาจเกิดขึ้นในการแก้ปัญหาทางคอมพิวเตอร์**

---

**1. ข้อจำกัดในการทำงานของคอมพิวเตอร์**

1. **การเก็บข้อมูล**:

- ข้อมูลที่คอมพิวเตอร์ใช้ในการแก้ปัญหาจำเป็นต้องมีความถูกต้องและครบถ้วน หากข้อมูลที่ได้รับมามีข้อบกพร่อง เช่น ไม่ครบถ้วน หรือผิดพลาด อาจทำให้การแก้ปัญหาไม่เป็นไปตามที่คาดหวัง.

2. **การให้เงื่อนไขข้อมูลที่จำกัด**:
    - คอมพิวเตอร์ไม่สามารถคิดแก้ปัญหาได้เองเหมือนมนุษย์ ต้องมีการตั้งโปรแกรมและให้เงื่อนไขที่ชัดเจนในทุกขั้นตอน.
3. **การวางคำสั่งที่จำกัด**:
    - การที่คำสั่งหรืออัลกอริทึมถูกกำหนดให้แก้ปัญหาได้ในขอบเขตที่กำหนดเท่านั้น ทำให้ไม่สามารถแก้ปัญหาที่อยู่นอกขอบเขตนี้ได้.

---

**2. ทุกคนมีปัญหามากมายที่ต้องแก้ไขในปัจจุบัน**

1. **บางคนไม่ให้ความสำคัญกับปัญหา**:
    - ผู้ใช้หรือผู้เกี่ยวข้องอาจไม่เข้าใจว่าแต่ละปัญหาสำคัญอย่างไร ทำให้ไม่สนใจหรือเพิกเฉย.
2. **ขาดการวิเคราะห์ปัญหาอย่างละเอียด**:
    - ปัญหาหลายอย่างเกิดจากการวิเคราะห์ที่ผิวเผิน ขาดการลงลึกในรายละเอียดของปัญหา หรือไม่มองปัญหาทั้งระบบ.
3. **การระบุปัญหาไม่ชัดเจน**:
    - เช่น ไม่สามารถบอกได้ว่าปัญหาเกิดขึ้นเพราะอะไร หรือการแก้ไขต้องเริ่มจากส่วนไหน.
4. **การสื่อสารที่ไม่เหมาะสม**:
    - หากผู้ใช้งานอธิบายปัญหาให้ผู้พัฒนาโปรแกรมไม่ชัดเจน หรือขาดการสื่อสารที่มีประสิทธิภาพ จะทำให้การแก้ปัญหาไม่ถูกต้อง.
5. **ขาดความรู้พื้นฐาน**:

- ผู้ใช้อาจไม่มีความรู้พื้นฐานเพียงพอในการอธิบายปัญหาที่เกิดขึ้น หรือไม่เข้าใจการทำงานของระบบ.

6. **พยายามแก้ปัญหาโดยไม่เข้าใจปัญหาจริง**:
    - แก้ไขแบบลองผิดลองถูกโดยไม่มีการวิเคราะห์ หรือแก้ปัญหาไม่ตรงจุด.

## 10. หน้า 41-42 Measurement and Test: Time and Test

```python
import time  # นำเข้าโมดูล time สำหรับจับเวลาการทำงานของโปรแกรม

def SequencesV1(n):  # นิยามฟังก์ชัน SequencesV1 เพื่อคำนวณผลรวมของตัวเลขตั้งแต่ 1 ถึง n
    total = 0  # กำหนดค่าเริ่มต้นของตัวแปร total เป็น 0
    for i in range(n):  # วนลูปตั้งแต่ 0 ถึง n-1
        total += (i + 1)  # เพิ่มค่า (i + 1) ไปยังตัวแปร total
    return total  # ส่งค่าผลรวมกลับไปยังผู้เรียกใช้ฟังก์ชัน

num = int(input("Input Value: "))  # รับค่าจำนวนเต็มจากผู้ใช้และเก็บไว้ในตัวแปร num
start = time.time()  # บันทึกเวลาที่เริ่มต้นการทำงานของฟังก์ชัน

print("ans = ", SequencesV1(num))  # เรียกใช้ฟังก์ชัน SequencesV1 และพิมพ์ผลลัพธ์
print("time = ", (time.time() - start) * 1000)
# คำนวณเวลาที่ใช้ในการทำงาน (เวลาปัจจุบัน - เวลาเริ่มต้น) และแปลงเป็นมิลลิวินาที
```

**อธิบายผลลัพธ์:**

1. Input Value: ผู้ใช้ระบุค่าอินพุต (ในตัวอย่างคือ 1000000).
2. ผลลัพธ์: แสดงผลรวมของตัวเลขตั้งแต่ 1 ถึง 1000000.
    - ผลลัพธ์: 500000500000
3. เวลา: แสดงเวลาที่ใช้ในการคำนวณ (เช่น 64.993... มิลลิวินาที).

HW1-CH1-Program ทอนเงิน

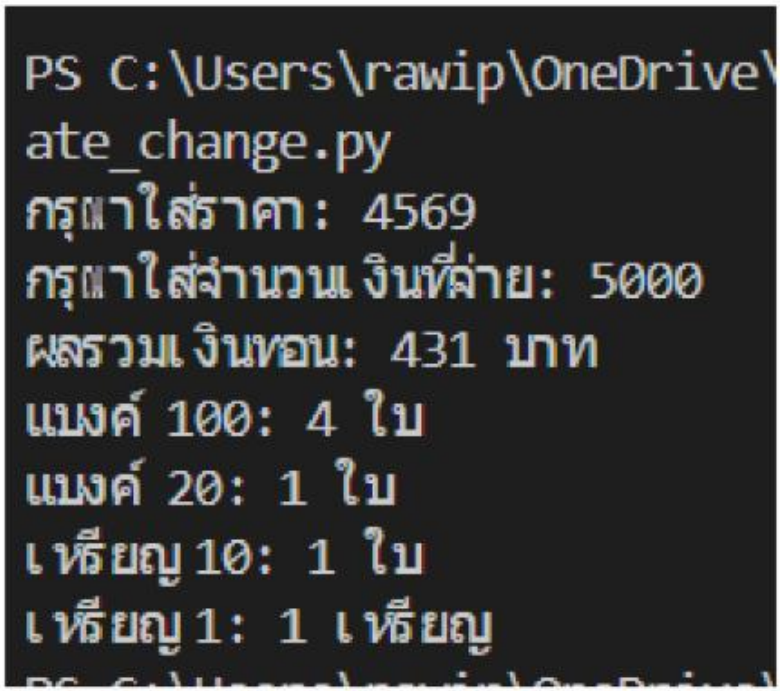
```python
def calculate_change(price, paid):
    # ฟังก์ชันนี้คำนวณเงินทอนจากราคาสินค้า (price) และเงินที่จ่าย (paid)

    if paid < price:  # ตรวจสอบว่าเงินที่จ่ายเพียงพอหรือไม่
        return "Error: เงินที่จ่ายไม่เพียงพอ"  # ส่งข้อความแจ้งเตือนกรณีเงินไม่พอ

    change = paid - price  # คำนวณเงินทอน
    denominations = [1000, 500, 100, 50, 20, 10, 5, 1]  # ประเภทของธนบัตรและเหรียญ
    denomination_names = {  # ชื่อของธนบัตรและเหรียญในภาษาไทย
        1000: 'แบงค์ 1000',
        500: 'แบงค์ 500',
        100: 'แบงค์ 100',
        50: 'แบงค์ 50',
        20: 'แบงค์ 20',
        10: 'เหรียญ 10',
        5: 'เหรียญ 5',
        1: 'เหรียญ 1'
    }

    result = {}  # เก็บผลลัพธ์ของจำนวนธนบัตร/เหรียญแต่ละประเภท
    total_change = 0  # เก็บผลรวมของเงินทอนทั้งหมด (ควรเท่ากับ change)

    for denom in denominations:  # วนลูปผ่านประเภทธนบัตร/เหรียญทั้งหมด
        count = change // denom  # คำนวณจำนวนธนบัตร/เหรียญที่ใช้
        if count > 0:  # ถ้าจำนวนธนบัตร/เหรียญที่ใช้มากกว่า 0
            if denom >= 10:  # ถ้าเป็นธนบัตร
                result[denomination_names[denom]] = f"{count} ใบ"
            else:  # ถ้าเป็นเหรียญ
                result[denomination_names[denom]] = f"{count} เหรียญ"
            total_change += count * denom  # เพิ่มมูลค่าของธนบัตร/เหรียญที่ใช้เข้าไปในผลรวมเงินทอน
        change %= denom  # หักจำนวนเงินที่ทอนแล้วออกจาก change

    return result, total_change  # คืนค่าผ[illegible]ต์เป็น tuple (result, total_change)
```

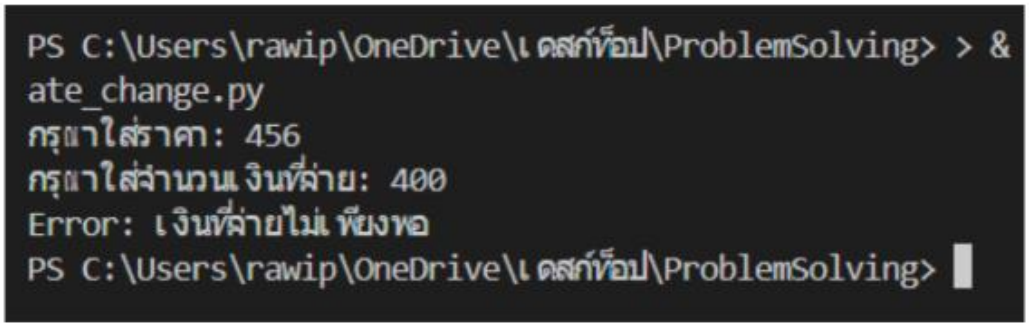
```
# รับข้อมูลจากผู้ใช้
price = int(input("กรุณาใส่ราคา: "))  # รับราคาสินค้าจากผู้ใช้
paid = int(input("กรุณาใส่จำนวนเงินที่จ่าย: "))  # รับจำนวนเงินที่ผู้ใช้จ่าย

# คำนวณการทอนเงิน
result = calculate_change(price, paid)

# ตรวจสอบผลลัพธ์ที่ได้รับ
if isinstance(result, tuple) and len(result) == 2:  # ตรวจสอบว่าผลลัพธ์เป็น tuple ที่มี 2 ค่า (re
    result, total_change = result  # แยกค่าผลลัพธ์ออกมา
    print(f"ผลรวมเงินทอน: {total_change} บาท")  # แสดงผลรวมเงินทอน
    for denomination, count in result.items():  # วนลูปผ่านผลลัพธ์แต่ละประเภทธนบัตร/เหรียญ
        print(f"{denomination}: {count}")  # แสดงจำนวนธนบัตร/เหรียญแต่ละประเภท
else:
    print(result)  # แสดงข้อความผิดพลาด (เงินไม่พอ)
```

---

## อธิบายโดยย่อ:

1. ฟังก์ชัน `calculate_change` คำนวณเงินทอนและแจกแจงธนบัตร/เหรียญที่ต้องใช้.
2. มีการตรวจสอบกรณีเงินที่จ่ายไม่พอ และคืนข้อความแสดงข้อผิดพลาด.
3. การคำนวณใช้ `//` (หารเอาจำนวนเต็ม) และ `%` (หารเอาเศษ) เพื่อจัดการเงินทอนในแต่ละขั้น.
4. มีการจัดการแยกประเภทธนบัตรและเหรียญ พร้อมแสดงผลในรูปแบบที่เข้าใจง่าย (เช่น ใบ, เหรียญ).




**การแก้ปัญหา 6 ขั้นตอนในบริบทของการคำนวณเงินทอน**

---

**1. ระบุปัญหาที่เกิดขึ้น (Identify the Problem)**

- **ปัญหา**: ต้องการเขียนโปรแกรมที่สามารถคำนวณเงินทอนจากจำนวนเงินที่จ่ายและราคาสินค้า พร้อมแจกแจงจำนวนธนบัตรและเหรียญที่ต้องใช้ในการทอน.

---

**2. ทำความเข้าใจกับปัญหา (Understand the Problem)**

- **ข้อมูลที่ต้องการ**:
    - price (ราคาสินค้า).
    - paid (จำนวนเงินที่จ่าย).
- **ผลลัพธ์ที่คาดหวัง**:
    - change (เงินทอน).
    - รายการธนบัตรและเหรียญที่ใช้ในการทอน.

---

**3. ระบุทางเลือกในการแก้ปัญหา (Identify Alternative Ways to Solve the Problem)**

- **วิธีที่ 1**: คำนวณเงินทอนโดยใช้การลบตรง (change = paid - price).
- **วิธีที่ 2**: ใช้ข้อมูลชนิดเงินสดที่ครอบคลุม (เช่น ธนบัตร: 1000, 500, 100 / เหรียญ: 10, 5, 1).
- **วิธีที่ 3**: ใช้การหารแบบโมดูลัส (change %= denom) เพื่อแจกแจงธนบัตร/เหรียญ.

---

**4. เลือกวิธีที่ดีที่สุดในการแก้ปัญหา (Select the Best Solution)**

- **เหตุผลในการเลือก**: วิธีที่ช่วยให้สามารถแจกแจงธนบัตรและเหรียญที่ใช้ได้ง่ายและรวดเร็ว เช่น ใช้การลบและโมดูลัสในการคำนวณ.

---

**5. ลำดับขั้นตอนในการแก้ปัญหา (List Instructions)**

1. รับข้อมูลจากผู้ใช้:
    - price (ราคา) และ paid (เงินที่จ่าย).
2. คำนวณเงินทอน (change = paid - price).

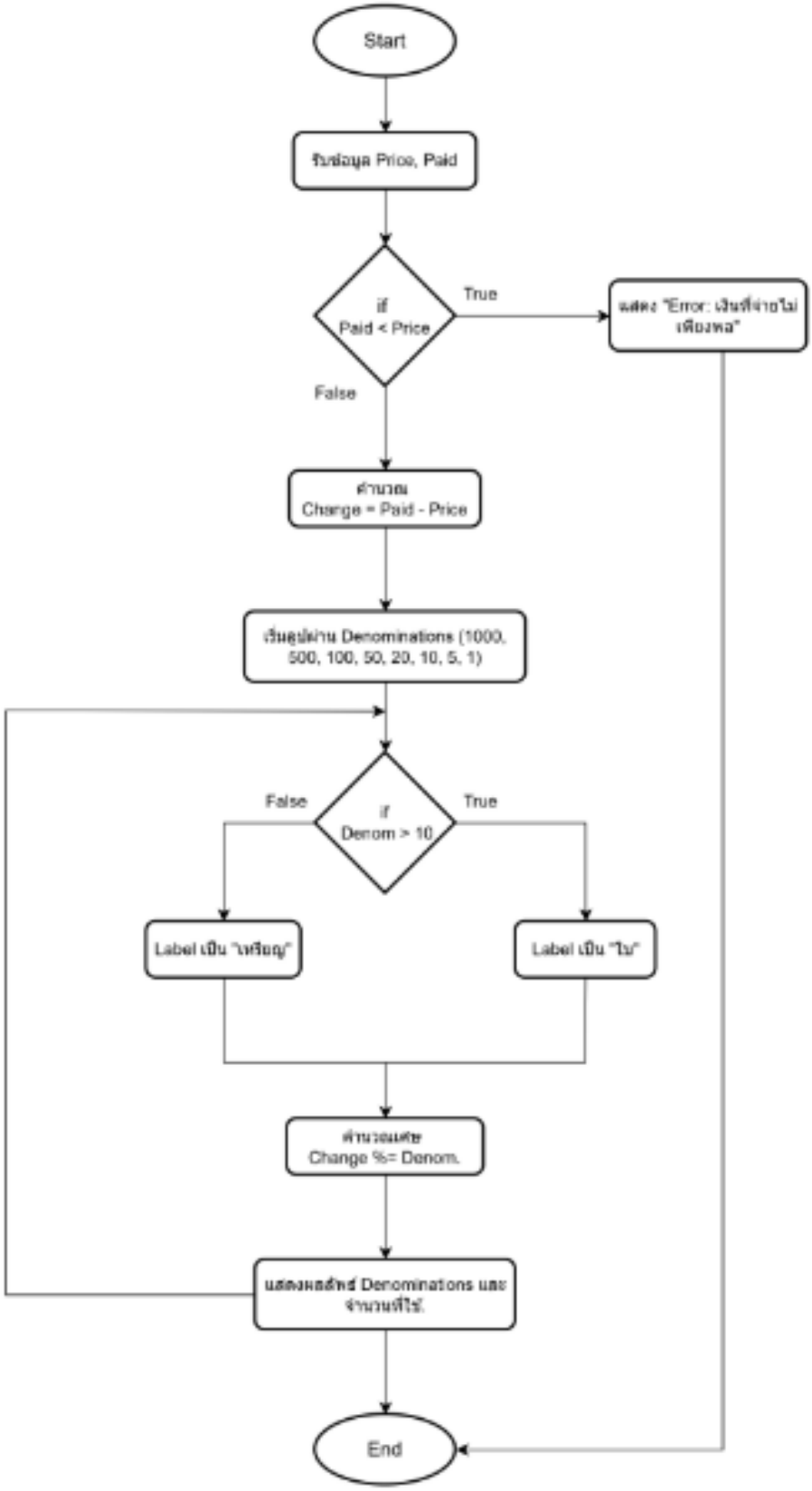
3. ตรวจสอบข้อผิดพลาด:
    - ถ้า paid < price ให้แสดงข้อความ "เงินที่จ่ายไม่พอ".
4. แจกแจงจำนวนธนบัตรและเหรียญ:
    - ใช้การหาร (//) และหารเอาเศษ (%) เพื่อคำนวณจำนวนเงินทอน.
5. แสดงผลลัพธ์.

---

**6. ประเมินวิธีการแก้ปัญหา (Evaluate the Solution)**

- **การทดสอบ**:
    - กรณีที่เงินจ่ายน้อยกว่าราคา: โปรแกรมต้องแสดงข้อความ "Error".
    - กรณีที่เงินจ่ายมากกว่าราคา: คำนวณเงินทอนและแจกแจงธนบัตร/เหรียญได้อย่างถูกต้อง.

---


Start
รับข้อมูล Price, Paid
if
Paid < Price
True
แสดง "Error: เงินที่จ่ายไม่เพียงพอ"
False
คำนวณ
Change = Paid - Price
เริ่มลูปผ่าน Denominations (1000, 500, 100, 50, 20, 10, 5, 1)
if
Denom > 10
False
True
Label เป็น "เหรียญ"
Label เป็น "ใบ"
คำนวณเศษ
Change %= Denom.
แสดงผลลัพธ์ Denominations และจำนวนที่ใช้.
End