โครงการแผยแพร่ผลงานวิชาการ
คณะอักษรศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ลำดับที่ 221

การเขียนโปรแกรมเพื่อประมวลผลภาษาธรรมชาติ

เรียนรู้การเขียนโค้ดเพื่อทำความเข้าใจข้อมูลภาษาและสร้างแอปพลิเคชันทางภาษาเบื้องต้น

รศ. ดร.อรรถพล ธำรงรัตนฤทธิ์

การเขียนโปรแกรมเพื่อประมวลผลภาษาธรรมชาติ

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

อรรถพล ธำรงรัตนฤทธิ์.

การเขียนโปรแกรมเพื่อประมวลผลภาษาธรรมชาติ.—กรุงเทพฯ : โครงการเผยแพร่ผลงานวิชาการ คณะอักษรศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2569.
430 หน้า.

1.การเขียนโปรแกรมคอมพิวเตอร์. 2. ภาษาคอมพิวเตอร์. I. ชื่อเรื่อง.

005.133

ISBN 978-616-621-084-2

การเขียนโปรแกรมเพื่อประมวลผลภาษาธรรมชาติ
ผู้เขียน : อรรถพล ธำรงรัตนฤทธิ์
ออกแบบปก : ปุณยนุช ไมตรีนุกุล
พิสูจน์อักษร : ปุณยนุช ไมตรีนุกุล และธนนท์ หลีน้อย
จัดรูปเล่ม : ปุณยนุช ไมตรีนุกุล

จัดพิมพ์โดยและจัดจำหน่ายโดย โครงการเผยแพร่ผลงานวิชาการ
คณะอักษรศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

พิมพ์ครั้งที่ 1 มีนาคม 2569 จำนวน 200 เล่ม

พิมพ์ที่ สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย

สารบัญ

สารบัญตาราง

บทที่ 1

บทที่ 2

บทที่ 3

บทที่ 4

บทที่ 5

บทที่ 7

บทที่ 8

บทที่ 9

บทที่ 10

สารบัญรูปภาพ

บทที่ 1

บทที่ 2

บทที่ 5

บทที่ 7

บทที่ 8

บทนำ

พรสวรรค์ของมนุษย์คือภาษา ภาษาเป็นเครื่องมือที่ทำให้มนุษย์สื่อสาร และแสดงออกความคิดที่ความซับซ้อนออกมาได้อย่างไม่มีที่สิ้นสุด การสื่อสารก่อให้เกิดการร่วมมือกันของสังคมทั้งด้านวิทยาศาสตร์ สังคม วัฒนธรรม ซึ่งเป็นปัจจัยที่สำคัญที่สุดปัจจัยหนึ่งที่ขับเคลื่อนให้มนุษยชาติพัฒนาต่อไปข้างหน้า จนมาถึงปัจจุบันนี้มนุษย์พัฒนาภาษาคอมพิวเตอร์ขึ้นเพื่อใช้สื่อสารกับคอมพิวเตอร์ซึ่งมีความสามารถเหนือมนุษย์ในการคำนวณทั้งในด้านความเร็ว และในด้านความแม่นยำ นักเขียนโปรแกรมเป็นผู้อยู่เบื้องหลังที่ใช้ภาษาคอมพิวเตอร์ในการติดต่อสื่อสารกับคอมพิวเตอร์ สั่งให้ทำงานต่าง ๆ แทนมนุษย์ได้หลายอย่าง เช่น ส่งข้อความหากันผ่านมือถือ ค้นคว้าหาข้อมูลจากแหล่งข้อมูลขนาดมหาศาล หรือแปลภาษาที่ไม่รู้จัก ภาษาคอมพิวเตอร์อยู่ในชีวิตประจำวันของเราจนเราไม่สามารถจินตนาการได้อีกแล้วว่า ถ้าหากวันหนึ่งเราไม่มีโปรแกรมเหล่านี้แล้วชีวิตจะดำเนินต่อไปได้อย่างไร การเขียนโปรแกรมจึงนับเป็นทักษะที่สำคัญที่สุดทักษะหนึ่งในปัจจุบันนี้ เป็นทักษะที่ทำให้เราสามารถติดต่อสื่อสาร และสั่งให้เครื่องคอมพิวเตอร์คำนวณและประมวลผลข้อมูลให้เราได้

ภาษาคอมพิวเตอร์ถูกออกแบบมาให้สามารถประยุกต์ใช้กับงานต่าง ๆ ได้อย่างกว้างขวาง ซึ่งเมื่อเขียนโปรแกรมได้คล่องแล้ว ทำให้นำไปเขียนเป็นโปรแกรมได้หลากหลายมาก ความกว้างของภาษาคอมพิวเตอร์เป็นความท้าทายอีกอย่างหนึ่งของผู้ที่หัดเขียนโปรแกรม และรายวิชาและหนังสือตำราเขียนโปรแกรมมักจะครอบคลุมเนื้อหาอย่างครบถ้วน โดยเฉพาะเนื้อหาที่เป็นพื้นฐานของวิทยาการคอมพิวเตอร์ เช่น เรื่องการวิเคราะห์อัลกอริทึม การเขียนโปรแกรมเชิงวัตถุขั้นสูง ซึ่งเป็นเรื่องที่น่าศึกษา แต่ว่าทำให้ผู้เรียนรู้สึกว่าเนื้อหาปริมาณมากเกินไป เข้าถึงได้ยาก ทำให้ผู้เรียนหมดกำลังใจในการศึกษาเรียนรู้ ในปัจจุบันการเขียนโปรแกรมไม่ใช่ทักษะสำหรับนักเขียนโปรแกรมแต่เพียงอย่างเดียว แต่เป็นทักษะที่ขาดไม่ได้สำหรับนักวิเคราะห์ข้อมูล (data analyst) นักวิทยาการข้อมูล (data scientist) นักวิจัย และอาชีพอื่น ๆ ที่จำเป็นต้องประมวลผลข้อมูลปริมาณมาก หรือทำให้กระบวนการทำงานต่าง ๆ อัตโนมัติมากขึ้น นั่นเองเป็นเหตุผลหลักที่ผู้เขียนต้องการเขียนหนังสือตำราที่สอนทักษะ และหัวข้อที่เฉพาะเจาะจงกับการวิเคราะห์ข้อมูล โดยเฉพาะอย่างยิ่งการวิเคราะห์ข้อมูลที่เป็นตัวอักษร หรือการประมวลผลภาษาธรรมชาติ ซึ่งคือการทำให้เครื่องคอมพิวเตอร์เข้าใจภาษา ทำให้สามารถทำความเข้าใจข้อมูลภาษาที่มีขนาดใหญ่ ๆ และสามารถประยุกต์ไปใช้สร้างเครื่องมือทางปัญญาประดิษฐ์ได้หลากลาย ผู้เขียนจึงคัดเลือกเนื้อหาที่เป็นแก่นสำคัญที่สุดในการเขียนโปรแกรมเพื่อวิเคราะห์ข้อมูลและประมวลภาษาธรรมชาติ เพื่อให้หนังสือเล่มนี้เข้าถึงได้ง่ายมากขึ้น โดยไม่ทำให้ผู้ที่ต้องการศึกษาสับสนหรือรู้สึกว่าเนื้อหาล้นเกินที่จะเข้าถึงได้

ผู้เขียนได้รับแรงบันดาลใจในการเขียนหนังสือเล่มนี้จากการสอนรายวิชาการเขียนโปรแกรมเพื่อการประมวลผลภาษาธรรมชาติเบื้องต้น ให้แก่นิสิตวิชาเอกเทคโนโลยีภาษาและสารสนเทศ คณะอักษรศาสตร์ จุฬาฯ การเขียนโปรแกรมเป็นทักษะที่น้อยคนมักจะคิดว่าเป็นทักษะที่นักอักษรศาสตร์พึงมี หรือเป็นทักษะที่นิสิตอักษรศาสตร์มีความสามารถที่จะ

เรียนได้สำเร็จ ผู้เขียนเองเป็นคนหนึ่งที่เรียนรู้การเขียนโปรแกรมเมื่อเรียนอยู่ในระดับมหาวิทยาลัยแล้ว และไม่ได้เป็นผู้ที่มีพื้นฐานทางคณิตศาสตร์และวิทยาศาสตร์เกินระดับมัธยมศึกษาตอนต้นไปเท่าใดนัก คนทั่วไปมักจะคิดว่าการเขียนโปรแกรมจำเป็นต้องมีพื้นฐานทางด้านคณิตศาสตร์และวิทยาศาสตร์ที่ดี ผู้เขียนจึงเกิดความกลัวว่าจะสามารถเรียนได้สำเร็จหรือไม่ แต่ที่จริงแล้วแก่นสำคัญของการเขียนโปรแกรม คือ การคิดแบบเป็นระบบระเบียบเป็นขั้นตอน เห็นโจทย์แล้วแยกแยะได้ว่าจะต้องมีขั้นตอนอะไรบ้าง ขั้นตอนหนึ่งจะนำไปสู่อีกขั้นตอนหนึ่งได้อย่างไร ซึ่งเป็นวิธีการคิดที่ไม่ได้ถูกเน้นมากนักในศาสตร์ใดเลย เพราะฉะนั้นผู้เขียนจึงเชื่ออย่างยิ่งว่าทุกคนสามารถเขียนโปรแกรมได้ถ้าหากเริ่มต้นปูพื้นฐานการคิดเชิงคำนวณ (computational thinking) และแก้โจทย์การเขียนโปรแกรมที่ไล่เรียงจากง่ายไปยาก

หนังสือเล่มนี้ เป็นหนังสือที่เหมาะกับทั้งผู้ที่ไม่เคยมีประสบการณ์การเขียนโปรแกรมเลย และผู้ที่เขียนโปรแกรมได้มาก่อนแล้ว หนังสือเล่มนี้สามารถแบ่งได้ออกเป็นสองส่วนหลัก ส่วนแรกเน้นการปูพื้นฐานการเขียนโปรแกรมเบื้องต้นสำหรับผู้ที่ไม่เคยมีประสบการณ์การเขียนโปรแกรม เนื้อหาจะใกล้เคียงกับตำราสอนเขียนโปรแกรมภาษาไพทอนทั่วไป เพียงแต่ว่าหนังสือเล่มนี้จะให้ความสำคัญกับการใช้โครงสร้างข้อมูลที่จำเป็นต่อการเก็บข้อมูลภาษา ได้แก่ ลิสต์ ทูเปิล ดิกชันนารี โดยไม่ลงลึกถึงทฤษฎีและอัลกอริทึม จากนั้นส่วนที่สองของหนังสือเล่มนี้จะเริ่มเข้าสู่มโนทัศน์สำคัญของการประมวลผลภาษาธรรมชาติ ได้แก่ การทำความสะอาดข้อมูล การตัดคำ การตัดประโยค การแยกประเภทของคำ โดยใช้ไลบรารีภาษาไพทอน รวมถึงโมเดลที่ใช้การเรียนรู้ของเครื่องในการวิเคราะห์ภาษา เนื้อหาจะปิดท้ายด้วยการใช้โมเดลภาษาขนาดใหญ่ ซึ่งเป็นทักษะที่ทำให้แม้แต่ผู้เริ่มฝึกหัดเขียนโปรแกรมสามารถสร้างแอปพลิเคชันการประมวลผลภาษาธรรมชาติที่ซับซ้อนได้

ผู้เขียนขอขอบคุณคณะอักษรศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยเป็นอย่างยิ่งสำหรับทุนสนับสนุนในการจัดทำหนังสือในโครงการเผยแพร่งานวิชาการคณะอักษรศาสตร์ และอนุญาตให้ผู้เขียนลาจากภาระงานสอนเพื่ออุทิศเวลา 1 ปีในการแต่งหนังสือเล่มนี้ ขอขอบคุณผู้ช่วยวิจัยทุกท่าน ที่ช่วยให้คำติชม เสริมเติมเนื้อหา วาดภาพประกอบ และตรวจแก้ภาษาไทยที่ฟังไม่ค่อยเหมือนคนไทยหลาย ๆ จุด ให้ฟังดูเป็นภาษามนุษย์ได้ ขอขอบคุณอาจารย์ Dan Jurafsky อาจารย์ Chris Manning และอาจารย์ Nianwen Xue ผู้สอนความรู้ทุกอย่างเกี่ยวกับการประมวลผลภาษาธรรมชาติให้กับผู้เขียนตั้งแต่ปริญญาตรีถึงปริญญาเอก และทำให้ศาสตร์การประมวลผลภาษาธรรมชาติเป็นแพสชันของผู้เขียนอย่างขาดกันไม่ได้ สุดท้ายนี้ผู้เขียนต้องขอขอบคุณครอบครัว และเพื่อน ๆ ที่ให้กำลังใจผู้เขียนตลอดระยะเวลาที่เขียนหนังสือเล่มนี้ ที่ต้องขอบคุณที่สุดคือจอห์นที่อยู่เคียงข้างผู้เขียน รองรับอารมณ์ทั้งดีทั้งร้ายของผู้เขียนมาโดยตลอด หากปราศจากบุคคลเหล่านี้หนังสือเล่มนี้คงไม่สามารถสำเร็จเป็นรูปเล่มสมบูรณ์ โดยที่ผู้เขียนไม่เสียสติไปเสียก่อนได้

อรรถพล ธำรงรัตนฤทธิ์

บทที่ 1
การเขียนโปรแกรมและการคิดเชิงคำนวณ

จุดมุ่งหมายของบทนี้

- ผู้อ่านรู้จักวิธีการคิดเชิงคำนวณ และสามารถคิดเชิงคำนวณได้
- ผู้อ่านสามารถวิเคราะห์และแบ่งปัญหาขนาดใหญ่ออกเป็นส่วนย่อยที่จัดการได้ง่ายขึ้น
- ผู้อ่านเข้าใจหลักการเขียนฟังก์ชันที่สามารถนำกลับมาใช้ซ้ำได้
- ผู้อ่านเข้าใจแนวคิดของการใช้คำสั่งวนซ้ำ `for` และ `while`
- ผู้อ่านสามารถใช้คำสั่ง `if` และ `else` ร่วมกับค่าบูลีนและตัวดำเนินการตรรกะ เช่น `and` และ `or` ได้อย่างถูกต้อง

หลักสำคัญของการเขียนโปรแกรมมิได้อยู่ที่การจดจำคำสั่งทั้งหมด แต่เน้นที่การนำคำสั่งมาประยุกต์ใช้และประกอบกันอย่างเหมาะสมเพื่อให้ระบบสามารถปฏิบัติงานตามที่ผู้พัฒนาต้องการได้อย่างมีประสิทธิภาพ เมื่อได้รับโจทย์ เช่น การดึงข้อมูลความคิดเห็นจากโซเชียลมีเดียเพื่อนำมาวิเคราะห์ว่าผู้สมัครรับเลือกตั้งรายใดได้รับความสนใจจากสาธารณชนมากที่สุด หรือเขียนแอปพลิเคชันในการวิเคราะห์การใช้ภาษาของข้อความที่กำหนดให้ จะพบว่าโจทย์ดังกล่าวมีขอบเขตที่กว้างและซับซ้อน จนอาจไม่รู้ว่าจะเริ่มจากจุดไหนก่อน การแก้ปัญหาเชิงคำนวณ (computational thinking) เป็นหลักการสำคัญที่จะช่วยในการแก้โจทย์ลักษณะนี้ โดยมีองค์ประกอบหลัก 4 ประการ

การแก้ปัญหาเชิงคำนวณ (Computational Thinking) วิธีการคิดที่ใช้ในการแบ่งปัญหาออกเป็นส่วนย่อย หาแบบแผน คิดอย่างเป็นนามธรรม และออกแบบให้เป็นระบบ เพื่อให้สามารถแก้ไขปัญหาได้อย่างมีประสิทธิภาพ

1. การแบ่งปัญหาออกเป็นส่วนย่อย (decomposition) หมายถึง การวิเคราะห์และแบ่งแยกโจทย์ปัญหาออกเป็นส่วนย่อย ๆ เพื่อให้สามารถจัดการและแก้ไขได้ง่ายขึ้น ตัวอย่างเช่น หากเราต้องการประกอบตู้ลิ้นชักใส่เสื้อผ้า กระบวนการนี้สามารถแบ่งออกเป็นขั้นตอนต่าง ๆ ได้แก่ การประกอบลิ้นชัก การประกอบโครงตู้ และการติดตั้งขาตู้ จากนั้นนำแต่ละส่วนมาประกอบรวมกันเป็นตู้ลิ้นชักที่สมบูรณ์
2. การสังเกตหาแบบแผน (pattern recognition) หมายถึง การพิจารณาและค้นหาแบบแผนหรือรูปแบบที่เกิดขึ้นซ้ำ ๆ ในปัญหาย่อย ๆ ซึ่งจะช่วยให้สามารถแก้ไขปัญหาได้อย่างมีประสิทธิภาพ เช่น ในกระบวนการประกอบลิ้นชัก แม้ว่าแต่ละลิ้นชักจะมีขนาดแตกต่างกัน แต่ขั้นตอนการประกอบ เช่น การขันน็อตทั้งสี่ด้านและการติดตั้งมือจับ ยังคงมีรูปแบบที่เหมือนกัน
3. การคิดแบบนามธรรม (abstraction) หมายถึง การระบุแก่นแท้ของปัญหาหรือกระบวนการ โดยตัดรายละเอียดที่ไม่จำเป็นออกไป ตัวอย่างเช่น ลิ้นชักทั้งหมดในตลาดจะมีลักษณะพื้นฐานที่คล้ายกัน คือ เป็นกล่องสี่เหลี่ยม มีฝาด้านหน้าที่สามารถดึงออกได้ ส่วนความแตกต่าง เช่น ขนาด วัสดุ หรือรูปทรงของหูจับ ถือเป็นรายละเอียดที่ไม่เกี่ยวข้องกับแก่นของการประกอบลิ้นชัก

4. การออกแบบอัลกอริทึม (algorithm design) หมายถึง การสร้างขั้นตอนหรือกระบวนการในการแก้ปัญหาอย่างชัดเจนและเป็นระบบ เพื่อให้สามารถนำไปปฏิบัติได้ ตัวอย่างเช่น อัลกอริทึมสำหรับการประกอบตู้ลิ้นชักใส่เสื้อผ้าอาจประกอบด้วยขั้นตอนในการประกอบแต่ละชิ้นส่วนและรวมเป็นโครงสร้างที่สมบูรณ์

ตัวอย่างอัลกอริทึมสำหรับการประกอบตู้ลิ้นชักสำหรับใส่เสื้อผ้า ประกอบไปด้วยขั้นตอนดังนี้

1. ไล่ประกอบขาตู้ แล้วเอาพักไว้ก่อน
2. ประกอบโครงตู้ แล้วเอาพักไว้ก่อน
3. ประกอบผลลัพธ์จาก 1 และ 2
4. ถ้าตู้มีลิ้นชัก k อัน ไล่ประกอบตัวลิ้นชักจนครบ k อัน
5. สอดลิ้นชักทั้ง k อันเข้าไปในโครงตู้

จะสังเกตได้ว่าอัลกอริทึมดังกล่าวมีการแบ่งแยกโจทย์ออกเป็นส่วนย่อย ๆ เพื่อให้สามารถแก้ไขปัญหาได้ทีละส่วน นอกจากนี้ ยังมีการนำกระบวนการคิดแบบนามธรรมมาใช้ โดยที่อัลกอริทึมไม่จำเป็นต้องคำนึงถึงจำนวนหรือขนาดของลิ้นชักที่ต้องประกอบ ซึ่งทำให้อัลกอริทึมสามารถนำไปประยุกต์ใช้ได้กับลิ้นชักทุกประเภทและทุกขนาด

เฉกเช่นเดียวกับการเขียนโปรแกรม ทักษะการคิดเชิงคำนวณเป็นทักษะที่จำเป็นอย่างยิ่งต่อการเขียนโปรแกรม เพราะช่วยให้ผู้พัฒนาสามารถย่อยปัญหาที่ซับซ้อนให้จัดการได้ง่ายขึ้น การสังเกตแบบแผนร่วมและการคิดแบบนามธรรมช่วยให้มองเห็นโครงสร้างหลักของปัญหาได้ชัดเจน และสามารถออกแบบขั้นตอนการทำงานได้อย่างเป็นระบบ จึงสามารถสร้างโปรแกรมที่มีความถูกต้อง ยืดหยุ่น และสามารถนำกลับมาใช้ซ้ำได้ในหลายบริบท เหมือนกับอัลกอริทึมการประกอบลิ้นชักที่สามารถประยุกต์ใช้ได้กับลิ้นชักหลากหลายรูปแบบ โปรแกรมที่ดีเองก็ควรมีความยืดหยุ่นเช่นเดียวกัน

ภาษาโปรแกรม ภาษาที่ใช้สื่อสารคำสั่งกับเครื่องคอมพิวเตอร์

ไพทอน (Python) เป็นชื่อภาษาโปรแกรม (programming language) เป็นภาษาที่ได้รับความนิยมที่สุดในการประมวลผลและวิเคราะห์ข้อมูลต่าง ๆ ในปัจจุบัน ไม่ว่าจะเป็นข้อมูลที่เป็นตัวเลข ข้อมูลที่อยู่ในฐานข้อมูลเรียบร้อยแล้ว หรือข้อมูลที่เป็นภาษา โปรแกรมชิ้นหนึ่งจะประกอบด้วยชุดคำสั่งที่ผู้เขียนโปรแกรมจะเขียนขึ้นเป็นภาษาไพทอน

ในบทนี้ผู้เรียนและผู้เริ่มต้นเขียนโปรแกรมจะได้เรียนรู้การติดตั้งและเริ่มต้นใช้ภาษาไพทอนบนคอมพิวเตอร์ และทักษะการคิดเชิงคำนวณ โดยจะสอดแทรกความรู้เรื่องการเขียนโปรแกรม เพื่อเป็นการปูพื้นฐานแนวคิดและเตรียมพร้อมอุปกรณ์ให้ผู้เรียนสามารถเรียนและทดลองปฏิบัติตามเนื้อหาและแบบฝึกหัดในบทที่ตามมาได้อย่างราบรื่น

เริ่มต้นกับภาษาไพทอน

การติดตั้งโปรแกรม Anaconda

สิ่งแรกที่ต้องทำก่อนเริ่มเขียนโปรแกรมคือการติดตั้งเครื่องมือที่จะใช้เขียนโปรแกรมส่วนประกอบต่าง ๆ ที่จำเป็นสำหรับการเขียนโปรแกรมในภาษาไพทอน สามารถดาวน์โหลดพร้อมกันได้ผ่านการลงโปรแกรม Anaconda ซึ่งมีเครื่องมือต่าง ๆ ดังต่อไปนี้

- ตัวแปลคำสั่ง (interpreter) สำหรับที่ภาษาไพทอน ทำหน้าที่แปลภาษาไพทอนให้เป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจอีกทีหนึ่ง

  ทั้งนี้ควรสังเกตเวอร์ชันของตัวแปลคำสั่งด้วย โดยควรให้เป็นเวอร์ชันล่าสุด เพื่อให้สามารถใช้คุณสมบัติที่พัฒนามาล่าสุด

- ไลบรารี (library) คือกลุ่มของโค้ดที่นักพัฒนาโปรแกรม หรือผู้เขียนโปรแกรมมักใช้บ่อย ไลบรารีภาษาไพทอนหลาย ๆ ตัวที่มักถูกนำมาใช้สำหรับการประมวลผลและวิเคราะห์ข้อมูล และมักถูกติดตั้งมาให้พร้อมใช้งานเมื่อทำการติดตั้ง Anaconda ทำให้สะดวกสบายในการลงโปรแกรม

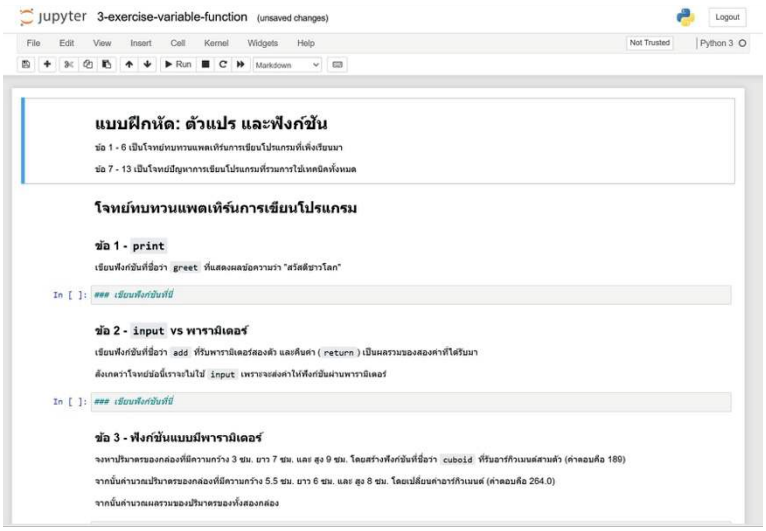
- Anaconda prompt คือ เทอร์มินัลประเภทหนึ่ง ซึ่งมักจะประกอบด้วยหน้าจอสีดำและข้อความที่แสดงผลให้ผู้ใช้พิมพ์คำสั่งลงไปบน Anaconda prompt ลงไปโดยตรง ผู้เขียนโปรแกรมสามารถใช้เทอร์มินัลในการรันคำสั่งติดตั้งไลบรารีต่าง ๆ และรันไฟล์โค้ดไพทอน

- สิ่งแวดล้อมสำหรับการพัฒนาแบบเบ็ดเสร็จ (Interactive Development Environment หรือ IDE) เป็นเครื่องมือที่มีไว้สำหรับอำนวยความสะดวกในการเขียนโค้ด ส่วนหลักคือหน้าต่างที่มีไว้สำหรับพิมพ์โค้ด และบันทึกโค้ดเก็บลงไฟล์ รวมถึงฟีเจอร์อื่น ๆ ที่ช่วยให้สามารถเขียนโค้ดได้อย่างมีประสิทธิภาพมากขึ้น เช่น ปรับเปลี่ยนสีตัวอักษรต่าง ๆ เพื่อให้ดูง่ายขึ้น และสามารถปรับหน้าต่างเพื่อแสดงโค้ดหลาย ๆ ไฟล์ หรือหลาย ๆ หน้า ได้ตามความพอใจ

ตัวแปลคำสั่ง (interpreter) มีหน้าที่แปลคำสั่งจากภาษาโปรแกรมมิ่ง (programming language) เป็นภาษาเครื่อง (machine code) เพื่อทำให้เครื่องเข้าใจและกระทำการ (execute) ตามคำสั่งที่ป้อนเข้าไป

ไลบรารี (library) โค้ดที่คนอื่นเขียนขึ้นมาให้แล้วพร้อมสำหรับใช้เป็นส่วนหนึ่งของโค้ดเราได้ทันที

เทอร์มินัล (Terminal) เป็นโปรแกรมที่ทำให้เราสั่งคอมพิวเตอร์ให้ทำงานต่าง ๆ ได้โดยการพิมพ์คำสั่งและกดปุ่ม Enter



ภาพที่ 1.1
ภาพตัวอย่างหน้าต่าง Jupyter Notebook

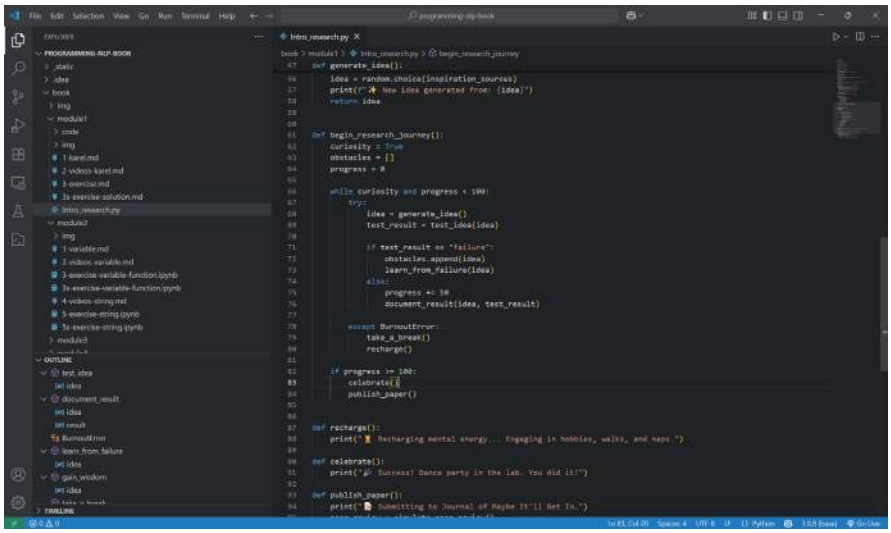
- Jupyter Notebook เป็น IDE ที่เรียบง่ายที่สุด และเป็นที่นิยมเป็นอย่างมากในหมู่นักวิเคราะห์ข้อมูล เพราะถูกออกแบบมาเพื่อการวิเคราะห์ข้อมูลโดยเฉพาะ เนื่องจาก Jupyter Notebook รองรับการรันโค้ดทีละก้อนเล็ก ๆ และแสดงผลการคำนวณออกมาให้เห็นชัดเจน IDE นี้จึงถูกเรียกว่า notebook เพราะเปรียบเสมือนกับนักวิทยาศาสตร์ข้อมูลทำการวิเคราะห์ข้อมูลและจดบันทึกกระบวนการวิเคราะห์และผลวิเคราะห์ไว้ในสมุด เพื่อให้ทราบที่มาที่ไปของผลสรุปต่าง ๆ ที่ได้จากการวิเคราะห์ ไฟล์ที่จะสามารถเปิดด้วย Jupyter Notebook ได้นั้นจะต้องมีนามสกุลไฟล์ (file extension) เป็น .ipynb

นามสกุลไฟล์ (file extension) นามสกุลที่ปรากฏในชื่อไฟล์ หรือประเภทไฟล์ เพื่อบ่งบอกว่าเป็นไฟล์ประเภทใด เช่น .txt .docx .jpg .json .py เป็นต้น

Visual Studio Code (VS Code)

Visual Studio Code (VS Code) นี้เป็น IDE ที่มีฟีเจอร์หลากหลายชนิด สามารถปรับรูปลักษณ์ตามความชอบได้หลายแบบ รวมถึงรองรับภาษาโปรแกรมหลายภาษา เหมาะกับการเขียนโปรแกรมที่ประกอบด้วยหลาย ๆ ไฟล์ และต้องมีการตรวจสอบโค้ดอย่างรัดกุม เป็น IDE ที่ดาวน์โหลดได้โดยไม่มีค่าใช้จ่าย และเป็นที่แพร่หลายมากในการเขียนโปรแกรมภาษาไพทอน นอกจากนี้ VS Code ยังสามารถเปิด Terminal ขึ้นมาใช้ในโปรแกรมได้อีกด้วย

หลังจากที่ผู้เขียนโปรแกรมได้ลงโปรแกรม Anaconda เรียบร้อยแล้ว ให้ผู้เขียนโปรแกรมลงโปรแกรม IDE ที่มีชื่อว่า Visual Studio Code (VS Code) โดยสามารถค้นหาและดาวน์โหลดได้จากเครื่องมือค้นหาบนอินเทอร์เน็ต



ภาพที่ 1.2
ภาพตัวอย่างหน้าต่าง
Visual Studio Code

หลังจากติดตั้งเรียบร้อยแล้ว ให้ไปที่ View > Extension จากนั้นพิมพ์ python และติดตั้ง python extension โดยควรสังเกตและติดตั้ง Extension ที่เป็นของ Microsoft

ทั้งนี้หน้าตัวอย่างแสดงผลของ VS Code ที่เปิดใน MacOS และ Windows นั้นอาจมีลักษณะแตกต่างกันบ้าง

การเขียนโปรแกรมแบบตอบโต้ (interactive)

หลังจากที่ลงโปรแกรม Anaconda เรียบร้อยแล้ว ผู้เขียนโปรแกรมสามารถเริ่มเขียนโปรแกรมเป็นภาษาไพทอนได้ทันที เพื่อทดสอบว่าการลงโปรแกรมนั้นเสร็จสมบูรณ์ดีแล้ว โดยควรเริ่มจากการเปิด Anaconda Prompt หรือ Anaconda Powershell อันเป็นเทอร์มินัลสำหรับผู้ที่ใช้ Windows หรือเปิด Terminal App ซึ่งเป็นเทอร์มินัลสำหรับผู้ที่ใช้ Mac ขึ้นมา จากนั้นให้พิมพ์คำว่า `ipython` ลงไปเพื่อเปิด Interactive Python Interpreter (ipython) และลองพิมพ์

```
print('Hello, world!')
```

และกด Enter เพื่อทำการส่งคำสั่งที่เขียนเป็นภาษาไพทอนเพื่อให้ตัวแปลภาษาไพทอนแปลเป็นภาษาเครื่องคอมพิวเตอร์และรันคำสั่ง จากนั้นหน้าจอจะแสดงผลดังนี้

```
Python 3.7.6 (default, Jan 8 2020, 13:42:34)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.29.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]: print('Hello, world!')
Hello, world!
In [2]::
```

โปรแกรม `ipython` เป็นโปรแกรมที่ทำให้ผู้เขียนโปรแกรมสามารถพิมพ์คำสั่งภาษาไพทอนและกดรันเพื่อเห็นผลลัพธ์ได้ทันที ทำให้ผู้ใช้และนักพัฒนาโปรแกรมสามารถตอบโต้กับเครื่องไปมาผ่านภาษาไพทอนได้

วิธีการเขียนโปรแกรมใส่ไฟล์ และสั่งรันทั้งไฟล์

Editor โปรแกรมที่ใช้สำหรับเปิดแก้ไขไฟล์

ในบางครั้งผู้เขียนโปรแกรมอาจต้องการเขียนโปรแกรมที่ค่อนข้างยาว และต้องการรันโปรแกรมทั้งหมดที่อยู่ในไฟล์ โดยผู้เขียนโปรแกรมสามารถเขียนโปรแกรมได้ โดยการเขียนคำสั่งทั้งหมดใส่ไว้ในไฟล์ก่อน ซึ่งจะต้องมีสกุลไฟล์ `.py` จากนั้นจึงสั่งให้ตัวแปลภาษาไพทอนแปลและรันคำสั่งทั้งหมดในครั้งเดียว โดยไล่ไปทีละบรรทัด ในหัวข้อนี้จะสอนวิธีการสั่งรันไฟล์ผ่านทาง VS Code

วิธีการรันโปรแกรมด้วย VS Code

วิธีการเขียนโปรแกรมใส่ไฟล์และรันโปรแกรมที่อยู่ในไฟล์นั้นผ่าน VS Code มีดังนี้

1. เปิด VS Code
2. กด File -> New File และตั้งชื่อว่า `myfirstprogram.py` ไฟล์จะถูกเปิดขึ้นบนหน้าต่างใหม่
3. พิมพ์คำว่า `print('Hello, world!')` ใส่ในไฟล์และกด Save
4. กด Terminal -> New Terminal เพื่อเปิด Terminal เป็นหน้าต่างใหม่บน VS Code
5. พิมพ์คำว่า `python myfirstprogram.py` ลงไปบนเทอร์มินัล จากนั้นให้กด enter

หน้าต่างเทอร์มินัลจะแสดงคำว่า `Hello, world!` อย่างไรก็ตาม หน้าตาเทอร์มินัลของแต่ละเครื่องก็จะแตกต่างกันไป

หุ่นยนต์คาเรล

คาเรล (Karel)

ทักษะการคิดเชิงคำนวณตามตัวอย่างที่กล่าวมาบทนำ ถือเป็นทักษะที่สำคัญยิ่งสำหรับการเขียนโปรแกรม ซึ่งหนึ่งในวิธีที่มีประสิทธิภาพและเป็นการเริ่มต้นที่ดีสำหรับการเริ่มฝึกทักษะดังกล่าวสำหรับผู้เริ่มต้น คือ การฝึกฝนผ่านโปรแกรมคาเรล (Karel) (Pattis, 1994) คาเรลเป็นหุ่นยนต์จำลองขนาดเล็กที่สามารถเคลื่อนที่ไปมาภายในโลกสี่เหลี่ยมที่ถูกกำหนดไว้ โดยผู้ใช้สามารถเขียนโปรแกรมด้วยภาษาไพทอนเพื่อสั่งการคาเรลให้ปฏิบัติตามคำสั่งต่าง ๆ ที่ตั้งไว้สำหรับการแก้โจทย์ในสถานการณ์ที่กำหนดได้

ในบทนี้ ผู้เรียนจะได้ฝึกฝนทักษะการคิดเชิงคำนวณเบื้องต้นผ่านโปรแกรมคาเรล เพื่อเป็นการวางรากฐานและเสริมสร้างทักษะการคิดเชิงคำนวณ โดยไม่ต้องจดจำคำสั่งจำนวนมาก ก่อนเข้าสู่เนื้อหาการเขียนโปรแกรมภาษาไพทอนที่ซับซ้อนขึ้นในบทถัด ๆ ไป

ข้อผิดพลาด (error)

ในโปรแกรมนี้ หุ่นยนต์คาเรลสามารถเคลื่อนที่ไปข้างหน้าได้ครั้งละหนึ่งก้าว แต่ไม่สามารถทะลุผ่านกำแพงได้ หากคาเรลชนกับกำแพงจะทำให้เกิดข้อผิดพลาด (error) และโปรแกรมจะหยุดทำงานทันที โจทย์ในโลกของคาเรลจะเริ่มจากการกำหนดจุดเริ่มต้นของ

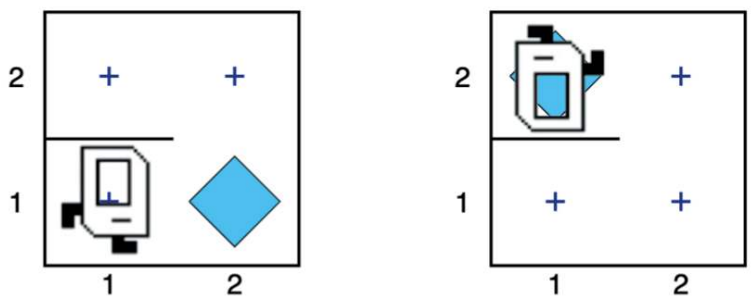
คาเรล ขนาดของโลกที่คาเรลอยู่ รวมถึงตำแหน่งของกำแพงและตำแหน่งของกระดิ่ง (beeper) คาเรลสามารถเก็บหรือวางกระดิ่งในจุดต่าง ๆ ได้ แต่หากพยายามเก็บกระดิ่งในจุดที่ไม่มีกระดิ่งวางอยู่ จะทำให้โปรแกรมเกิดข้อผิดพลาดขึ้นเช่นกัน คำสั่งที่ใช้ในการแก้ปัญหาในโจทย์คาเรลประกอบด้วยคำสั่งดังต่อไปนี้

ตารางที่ 1.1
คำสั่งพื้นฐานในโปรแกรมคาเรล

คำสั่ง	คำอธิบาย
`move()`	สั่งให้คาเรลเดินไปข้างหน้า หากข้างหน้ามีกำแพง คาเรลจะชนกำแพงและทำให้โปรแกรมเกิดข้อผิดพลาด
`turn_left()`	สั่งให้คาเรลเลี้ยวซ้าย
`put_beeper()`	วางกระดิ่งหนึ่งอันไว้ตรงจุดที่ยืนอยู่
`pick_beeper()`	เก็บกระดิ่งขึ้นมาหนึ่งอันจากตรงจุดที่ยืนอยู่ (ถ้าไม่มีกระดิ่งที่จุดนั้น จะทำให้โปรแกรมเกิดข้อผิดพลาด)

ตัวอย่าง

สมมติว่าคาเรลอยู่ในโลกขนาด 2x2 มีจุดเริ่มต้นอยู่ที่มุมซ้ายล่างของโลกและหันหน้าไปทางทิศตะวันออก และมีจุดหมายคือให้คาเรลเก็บกระดิ่งที่มุมขวาล่าง และไปเดินทางไปยังมุมซ้ายบน ดังภาพข้างล่าง



ภาพที่ 1.3
โลกของคาเรลในโจทย์ 1 จุดเริ่มต้นของโจทย์แสดงอยู่ในภาพด้านซ้าย จุดมุ่งหมายของโจทย์แสดงอยู่ในภาพด้านขวา

การแก้โจทย์นี้ ต้องอาศัยการนำคำสั่งหลาย ๆ คำสั่งมาต่อกันเป็นโปรแกรม (program) ในลำดับที่เหมาะสมเพื่อให้คาเรลสามารถเก็บกระดิ่งและเดินทางไปยังจุดที่โจทย์กำหนดได้ ดังนี้

โปรแกรม (program)
ชุดคำสั่ง

```
move()
pick_beeper()
turn_left()
move()
turn_left()
move()
put_beeper()
```

ถ้าหากโปรแกรมสั่งให้คาเรลชนกำแพง ดังโค้ดข้างล่างนี้

```
move()
move()
```

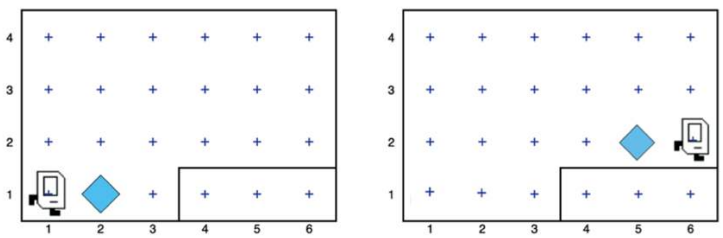
โปรแกรมจะเกิดข้อผิดพลาด และจะหยุดทำงานทันที หรือถ้าหากโปรแกรมสั่งให้คาเรลหยิบกระดิ่งในจุดที่ไม่มีกระดิ่ง ดังโค้ดข้างล่างนี้

```
move()
turn_left()
move()
pick_beeper()
```

โปรแกรมจะเกิดข้อผิดพลาด และจะหยุดทำงานทันทีเช่นกัน

โปรแกรมและฟังก์ชัน

หากว่าโจทย์ให้คาเรลไปหยิบกระดิ่งตรงหน้าเนินแล้วเดินข้ามเนินไปวางกระดิ่งที่คอลัมน์ที่ 5 ดังภาพที่ 1.2 ด้านล่าง



ภาพที่ 1.4 โลกของคาเรลในโจทย์ 2 จุดเริ่มต้นของโจทย์แสดงอยู่ในภาพด้านซ้าย จุดมุ่งหมายของโจทย์แสดงอยู่ในภาพด้านขวา

โจทย์นี้มีลักษณะที่ซับซ้อนมากขึ้นกว่าโจทย์แรก เนื่องจากต้องใช้คำสั่งหลายบรรทัดในการแก้โจทย์ โดยผู้เรียนจะต้องเขียนโค้ดภาษาไพทอนเพื่อสั่งการคาเรล โดยสามารถเขียนโค้ดไพทอนเก็บไว้ในไฟล์ `my_karel_first.py` ได้ดังนี้

```
# โปรแกรมสำหรับแก้โจทย์แรกในบทนี้
from stanfordkarel import *
def main():
    move()
    pick_beeper()
    move()
    turn_left()
    move()
    turn_left()
    turn_left()
    turn_left()
    move()
    move()
    put_beeper()
    move()
```

```
if __name__ == "__main__":
    run_karel_program()
```

ส่วนประกอบของโปรแกรมในไฟล์ `my_karel_first.py` นี้มีดังนี้

คอมเมนต์

```
# โปรแกรมสำหรับแก้โจทย์แรกในบทนี้
```

คอมเมนต์ (comment) ส่วนของโปรแกรมที่ไม่ถูกแปลงคำสั่ง

ข้อความที่นำหน้าด้วยเครื่องหมาย `#` เรียกว่า หมายเหตุ หรือคอมเมนต์ (comment) ซึ่งเป็นการแสดงความคิดเห็นหรือให้ข้อมูลเพิ่มเติมสำหรับผู้อ่านโค้ด โดยผู้เขียนโค้ดมักคอมเมนต์อธิบายการทำงานของโปรแกรมและจุดสำคัญที่ควรทราบ เพื่อให้ผู้ที่นำโค้ดไปใช้งานต่อสามารถเข้าใจการทำงานของโค้ดนั้นได้อย่างชัดเจน นอกจากนี้ การใช้คอมเมนต์ยังเป็นแนวทางในการช่วยเพิ่มความเข้าใจร่วมกันในทีมผู้พัฒนา และช่วยในกรณีที่ต้องกลับมาทบทวนโปรแกรมในภายหลัง

การนำเข้าไลบรารี

```
from stanfordkarel import *
```

ไลบรารี (library) คือ คลังของโค้ดหรือฟังก์ชันที่ถูกเขียนขึ้นล่วงหน้าและรวบรวมไว้เพื่อให้นักพัฒนาสามารถนำไปใช้งานได้โดยไม่ต้องเขียนโคดใหม่ตั้งแต่ต้น ไลบรารีมักจะประกอบไปด้วยฟังก์ชัน และเครื่องมือที่ช่วยในการทำงานเฉพาะด้าน ในบทนี้ไลบรารีที่ต้องนำเข้ามาคือ `stanfordkarel` ซึ่งในไลบรารีนี้ประกอบด้วยฟังก์ชันที่ใช้ในการสั่งการคาเรลให้ปฏิบัติงานต่าง ๆ เช่น `move()` และ `put_beeper()`

การประกาศฟังก์ชัน

```
def main():
    move()
    pick_beeper()
    move()
```

ส่วนนี้เรียกว่า ฟังก์ชัน โดยฟังก์ชันมีลักษณะสำคัญดังนี้:

- ส่วนหัวของฟังก์ชัน (function header) ประกอบด้วย
    - คำสำคัญ (keyword) `def` ซึ่งใช้สำหรับประกาศฟังก์ชันใหม่
    - ชื่อของฟังก์ชันซึ่งตามหลังคำว่า `def` และปิดท้ายด้วยวงเล็บเปิด-ปิด และโคลอน `():`
- ส่วนเนื้อหาของฟังก์ชัน (function body) คือ กลุ่มของคำสั่งที่ประกอบอยู่ภายในฟังก์ชัน โดยคำสั่งเหล่านี้จะต้องมีการย่อหน้า (indentation) ในทุกบรรทัด เพื่อบ่งชี้ว่าคำสั่งเหล่านี้เป็นส่วนหนึ่งของฟังก์ชันนั้น

การกำหนดคำสั่งการรันไฟล์ไพทอน

```
if __name__ == "__main__":
 run_karel_program()
```

รัน (run) การเรียกใช้งานโค้ดในโปรแกรม

โปรแกรมตัวอย่างนี้ถูกจัดเก็บในไฟล์ที่มีนามสกุล `.py` ซึ่งเมื่อไฟล์ถูกเรียกใช้ โปรแกรมจะทำงานทั้งไฟล์ โค้ดที่แสดงข้างต้นเป็นการกำหนดลำดับการทำงานของโปรแกรม โดยใช้โครงสร้าง `if __name__ == "__main__":` เพื่อระบุว่า หากโปรแกรมถูกรันโดยตรง ฟังก์ชัน `run_karel_program()` จะถูกเรียกใช้งานก่อน ซึ่งฟังก์ชันนี้มีหน้าที่ในการโหลดหน้าต่างที่แสดงผลหุ่นยนต์คาเรล จากนั้นโปรแกรมจะไปเรียกฟังก์ชัน `main()` ซึ่งได้ถูกประกาศไว้ในไฟล์เดียวกันนี้

การแบ่งปัญหาออกเป็นส่วนย่อย

สิ่งแรกที่ผู้เรียนสามารถสังเกตได้จากโค้ด `my_karel_first.py` คือ โค้ดนั้นมีความยาวและเข้าใจยาก ในการเขียนโปรแกรมนั้น ผู้เขียนโปรแกรมจำเป็นต้องวาดแผนภาพและตรวจสอบการทำงานของโค้ดแต่ละส่วนเพื่อทำความเข้าใจ ซึ่งการที่โค้ดมีลักษณะที่ยาวและเข้าใจยากนั้น อาจส่งผลให้ปรับแก้โค้ดได้ยาก โค้ด `my_karel_first.py` นี้จึงนับว่าไม่มีประสิทธิภาพเท่าที่ควร ซึ่งในการแก้ปัญหานี้ ผู้เขียนโปรแกรมสามารถสร้างฟังก์ชันใหม่ขึ้นมาเพื่อปรับปรุงโครงสร้างของโค้ดให้มีความชัดเจนและสามารถเข้าใจได้ง่ายขึ้นได้ เช่น

```
turn_left()
turn_left()
turn_left()
```

คำสั่งนี้ แท้จริงแล้วเป็นคำสั่งที่สั่งคาเรลหันขวา โดยการสั่งให้หันซ้ายสามครั้ง ดังนั้นผู้เขียนโปรแกรมจึงสามารถสร้างฟังก์ชันใหม่ `turn_right()` ขึ้นมาเพื่อทำให้โค้ดอ่านง่ายขึ้นดังนี้

```
def main():
    move()
    pick_beeper()
    move()
    turn_left()
    move()
    turn_right() # หันขวา ฟังก์ชันใหม่ move()
    move()
    put_beeper()
    move()
def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

การสร้างฟังก์ชัน (function definition / declaration) คือ เขียนฟังก์ชันขึ้นมาใหม่และตั้งชื่อของฟังก์ชันเพื่อให้พร้อมสำหรับการนำไปใช้

การเรียกฟังก์ชัน (function call) คือ การนำฟังก์ชันที่ตัวเราได้สร้างไว้เรียบร้อยแล้วมาใช้

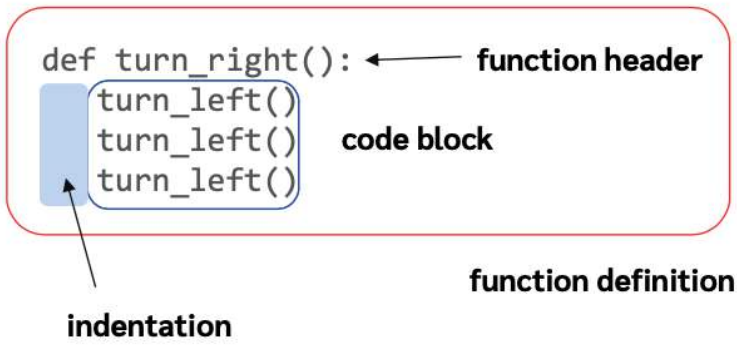
เพื่อปรับปรุงโครงสร้างของโค้ด ในโค้ดดังกล่าวได้มีการสร้างฟังก์ชันใหม่ชื่อ `turn_right()` ซึ่งมีบทบาทสำคัญในการทำให้โค้ดในฟังก์ชัน `main()` มีความกระชับและชัดเจนมากขึ้น การย้ายคำสั่งออกจากฟังก์ชัน `main()` ไปไว้ในฟังก์ชันย่อยนี้ไม่เพียงแต่ช่วยลดปริมาณโค้ดใน

ฟังก์ชันหลัก แต่ยังช่วยให้สามารถดูแลรักษาและปรับปรุงโค้ดในระยะยาวได้ง่ายและมีประสิทธิภาพมากขึ้นเช่นกัน

หลังจากที่ฟังก์ชันใหม่ถูกสร้างขึ้น ผู้เขียนโปรแกรมสามารถเรียกใช้งานฟังก์ชันจากส่วนต่าง ๆ ของโปรแกรมได้โดยการพิมพ์ชื่อฟังก์ชันตามด้วยเครื่องหมายวงเล็บ `()` ซึ่งเป็นส่วนที่จำเป็นสำหรับการเรียกใช้งานฟังก์ชัน ตัวอย่างเช่น คำสั่ง `turn_right()` หรือฟังก์ชันพื้นฐานอื่น ๆ เช่น `move()` และ `pick_beeper()` เป็นคำสั่งที่ทำงานในลักษณะเดียวกัน แต่สำหรับสองฟังก์ชันหลังนี้ ผู้เขียนโปรแกรมไม่จำเป็นต้องสร้างขึ้นใหม่ เนื่องจากมีการนำเข้าจากไลบรารีที่เกี่ยวข้องแล้วในตอนต้น

สรุปส่วนต่าง ๆ ของฟังก์ชันได้ดังนี้

ภาพที่ 1.5
ส่วนประกอบของฟังก์ชัน



ข้อควรระวังในการเขียนฟังก์ชัน

การสร้างฟังก์ชันในภาษาไพทอนจำเป็นต้องปฏิบัติตามรูปแบบที่ถูกต้องตามกฎไวยากรณ์ (syntax) เนื่องจากภาษาไพทอน เป็นภาษาการเขียนโปรแกรมที่มีโครงสร้างชัดเจนและยึดกับไวยากรณ์อย่างเคร่งครัด การเขียนโค้ดที่มีลักษณะผิดจากไวยากรณ์แม้เพียงเล็กน้อยก็จะส่งผลให้โปรแกรมไม่สามารถทำงานได้ตามที่คาดหมาย ตัวแปลภาษาไพทอนจะตรวจสอบโค้ดเมื่อมีการรันโปรแกรม และหากพบความผิดพลาด โปรแกรมจะหยุดทำงานทันทีพร้อมแสดงข้อความแจ้งข้อผิดพลาด (error message) ซึ่งศัพท์เทคนิคในการเขียนโปรแกรมเรียกว่าโยน (throw) ข้อผิดพลาด เพื่อช่วยให้ผู้เขียนโปรแกรมสามารถตรวจสอบและแก้ไขโค้ดได้อย่างถูกต้อง ข้อผิดพลาดที่อาจเกิดขึ้นระหว่างเขียนโปรแกรมมีหลายประเภท ข้อผิดพลาดที่พบได้บ่อยประเภทหนึ่งคือ ข้อผิดพลาดทางไวยากรณ์ (syntax error) เช่น การไม่ได้ใส่วงเล็บ การพิมพ์ชื่อฟังก์ชันผิด หรือการจัดวางบรรทัดโค้ดที่ไม่ถูกต้อง เมื่อพบข้อผิดพลาดเช่นนี้ ตัวแปลภาษาไพทอนก็จะทำการโยน `SyntaxError` ดังตัวอย่างต่อไปนี้

ข้อผิดพลาด (error message) ข้อผิดพลาดที่อาจเกิดขึ้นในระหว่างการเขียนโปรแกรม โดยเครื่องจะแสดงผลข้อผิดพลาดขึ้นมาหากเกิดข้อผิดพลาด ข้อผิดพลาดมีหลายประเภท เช่น ข้อผิดพลาดทางไวยากรณ์ เป็นต้น

```
File "my_first_karel.py", line 4
 def main()
 ^
SyntaxError: invalid syntax
```

จากตัวอย่างข้างต้น ตัวแปลภาษาไพทอนโยน `SyntaxError` มา เพราะว่าผู้เขียนโปรแกรมไม่ได้ใส่ `:` หลังชื่อฟังก์ชัน ในบรรทัดที่ 4 การแจ้งเตือนนี้เพื่อแจ้งให้ทราบว่าผู้เขียนโปรแกรมนั้นเขียนโค้ดผิดไวยากรณ์

อีกตัวอย่างหนึ่งคือ ในกรณีที่ผู้เขียนโปรแกรมลืมใส่วงเล็บเปิดและปิด (`()`) ในท้ายชื่อฟังก์ชัน ก็จะเกิดข้อผิดพลาดทางไวยากรณ์ (`SyntaxError`) ขึ้นมาเช่นเดียวกัน

```
File "my_first_karel.py", line 4
 def main:
 ^
SyntaxError: invalid syntax
```

ข้อผิดพลาดอีกประเภทที่พบบ่อยสำหรับผู้ที่กำลังเริ่มต้นเขียนโปรแกรมคือ ข้อผิดพลาดในการจัดย่อหน้า (indentation) ซึ่งหากมีข้อผิดพลาดเกี่ยวกับการจัดย่อหน้า เช่น ย่อหน้าเกินหรือย่อหน้าขาด ตัวแปลภาษาไพทอนก็จะโยนข้อผิดพลาด `IndentationError` มาให้ ดังตัวอย่างดังนี้

```
File "my_first_karel.py", line 5
 move()
 ^
IndentationError: expected an indented block
```

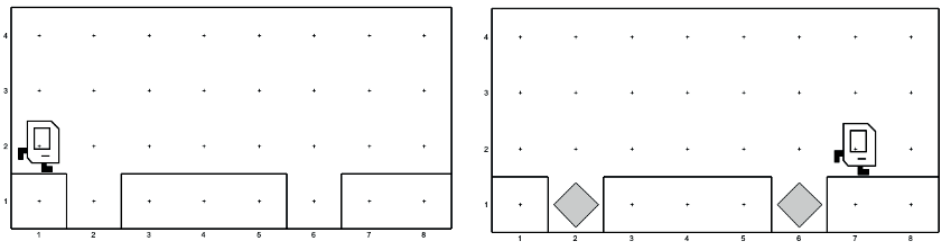
ตัวแปลภาษาไพทอนโยน `IndentationError` มาให้ถ้าเราลืมกั้นย่อหน้าให้คำสั่ง `move()` ในบรรทัดที่ 5 พร้อมแนะนำวิธีการแก้ว่าอาจจะต้องเพิ่มย่อหน้าให้เป็นบล็อกใหม่ (*expected an indented block*)

เมื่อผู้เขียนโปรแกรมเรียนรู้การใช้คำสั่งใหม่ ๆ มากขึ้น ก็จะเผชิญกับข้อผิดพลาดประเภทต่าง ๆ มากขึ้นเช่นเดียวกัน ผู้เขียนโปรแกรมควรจะสามารถอ่านและทำความเข้าใจได้ว่าตัวแปลภาษาไพทอนนั้นพยายามชี้ให้เห็นถึงข้อผิดพลาดประเภทใด เช่น ข้อผิดพลาดทางไวยากรณ์ (`SyntaxError`) หรือข้อผิดพลาดในเรื่องของคีย์ (`KeyError`) และจะต้องแก้ไขอย่างไรเพื่อให้โปรแกรมสามารถทำงานได้อย่างถูกต้อง ซึ่งในบทต่อ ๆ ไปจะมีการกล่าวถึงข้อผิดพลาดประเภทอื่น ๆ เพิ่มเติม

การคิดเชิงคำนวณในการเขียนโปรแกรม

ในการแก้ปัญหาเกี่ยวกับการเขียนโปรแกรมทุกครั้ง ควรเริ่มต้นจากการแบ่งปัญหาออกเป็นส่วนย่อยก่อนเสมอ ซึ่งการแบ่งปัญหาเป็นส่วนย่อยจะช่วยลดความซับซ้อนของโจทย์ ทำให้ผู้เขียนโปรแกรมสามารถจัดการกับปัญหาแต่ละส่วนได้อย่างเป็นระบบ และยังช่วยให้สามารถทดสอบและตรวจสอบโปรแกรมแต่ละส่วนได้ง่ายและรวดเร็วขึ้น ทำให้สามารถระบุข้อผิดพลาดและปรับปรุงโปรแกรมได้อย่างมีประสิทธิภาพ

ตัวอย่างเช่น หากพิจารณาโจทย์ที่กำหนดให้คาเรลยืนอยู่บนถนนที่มีหลุมบ่อ ซึ่งผู้เขียนโปรแกรมจะต้องวางกระดิ่งลงในหลุมเพื่อทำให้ถนนเรียบและสามารถเดินต่อไปได้ หากตั้งว่าโลกที่คาเรลเริ่มต้นอยู่นั้นเป็นไปตามภาพ 1.3 ซ้ายล่าง หลังจากที่ผู้เขียนโปรแกรมได้รันโปรแกรมเสร็จสมบูรณ์แล้ว ถนนจะมีลักษณะเป็นไปตามภาพขวาล่าง ในกระบวนการนี้ คาเรลต้องตัดสินใจและดำเนินการแก้ปัญหาด้วยการตรวจสอบตำแหน่งของหลุมบ่อ และวางกระดิ่งลงในตำแหน่งที่เหมาะสม ซึ่งลำดับขั้นตอนเหล่านี้เป็นการแก้ไขปัญหาอย่างเป็นระบบ



ภาพที่ 1.6
โลกของคาเรลในโจทย์ 4 จุดเริ่มต้นของโจทย์แสดงอยู่ในภาพด้านซ้าย จุดมุ่งหมายของโจทย์แสดงอยู่ในภาพด้านขวา

แท้จริงแล้ว โจทย์นี้ไม่ได้มีความซับซ้อนมากนัก แนวทางการแก้ปัญหาที่ได้มาโดยไม่อาศัยกระบวนการคิดเชิงคำนวณ จะมีลักษณะดังโค้ดด้านล่างนี้ ซึ่งในที่นี้จะหยิบยกขึ้นมาเฉพาะฟังก์ชัน `main()` และสมมติว่าผู้เขียนโปรแกรมได้ดำเนินการนำเข้าไลบรารีที่จำเป็นทั้งหมดและได้บันทึกโค้ดลงในไฟล์ที่มีนามสกุล `.py` แล้ว

```
def main():
    move()
    turn_right()
    move()
    put_beeper()
    turn_right()
    turn_right()
    move()
    turn_right()
    move()
    move()
    move()
    # ไม่สมบูรณ์ยังมีต่ออีก...
```

อย่างไรก็ตาม ควรตระหนักว่าการใช้กระบวนการคิดเชิงคำนวณในการออกแบบโปรแกรมเป็นสิ่งสำคัญที่จะช่วยให้การแก้ปัญหาเป็นไปอย่างมีประสิทธิภาพ โดยเฉพาะในกรณีที่ปัญหามีความซับซ้อนมากขึ้น ข้อสังเกตที่สำคัญคือ โค้ดที่เขียนขึ้นนั้นมีลักษณะที่ยาวและซับซ้อน ทำให้ต้องใช้เวลาในการทำความเข้าใจนาน จากตัวอย่าง หากพิจารณาโค้ดแต่ละบรรทัดโดยแยกออกจากบริบททั้งหมด จะไม่สามารถทราบได้อย่างแน่ชัดว่าคาเรลอยู่ที่ตำแหน่งใดในโลกเสมือน หรือกำลังหันหน้าไปในทิศทางใด การออกแบบโค้ดอย่างมีระบบและแบ่งส่วนการทำงานให้ชัดเจน จะช่วยให้โปรแกรมที่เขียนขึ้นเข้าใจง่ายและตรวจสอบได้ง่ายขึ้น

ในกรณีนี้ สามารถทดลองแก้โจทย์ 1.3 ใหม่อีกครั้ง โดยใช้วิธีการแบ่งปัญหาออกเป็นส่วนย่อย ซึ่งเป็นขั้นตอนแรกของกระบวนการคิดเชิงคำนวณ ทำให้การแก้ปัญหามีประสิทธิภาพและเป็นระบบมากขึ้น โดยโจทย์นี้สามารถแยกออกเป็น 3 ปัญหาย่อย ได้แก่

การแบ่งปัญหาออกเป็นส่วนย่อย เป็นขั้นตอนแรกของการคิดเชิงคำนวณ

1. การซ่อมหลุมแรก
2. การเดินไปยังหลุมที่สอง
3. การซ่อมหลุมที่สอง

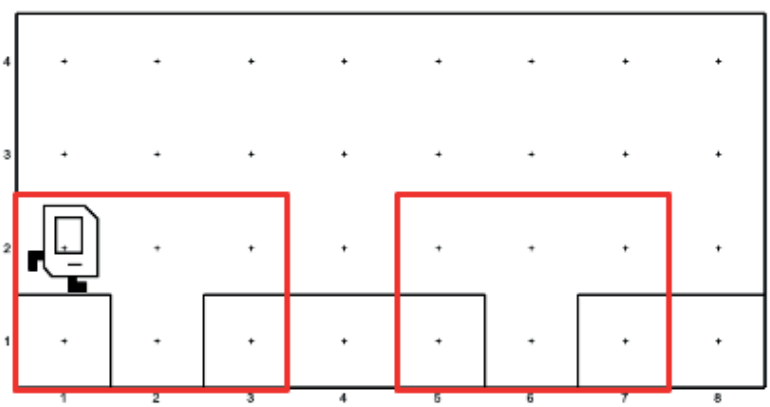
เพื่อแก้ปัญหาย่อยแต่ละข้อ ผู้เขียนโปรแกรมจะต้องสร้างฟังก์ชันเฉพาะสำหรับแต่ละปัญหา ซึ่งจะช่วยให้โค้ดมีความชัดเจนและเข้าใจง่ายขึ้น ฟังก์ชันเหล่านี้จะถูกเรียกใช้ตามลำดับที่กำหนด ทำให้สามารถแก้ปัญหาได้อย่างเป็นระบบและตรวจสอบได้ง่าย

```
def fill_hole1():
    pass

def move_to_next_hole()
    move()
    move()

def fill_hole2():
    pass
```

คำสั่ง `pass` ใช้เพื่อเว้นตำแหน่งไว้สำหรับการเขียนโค้ดในฟังก์ชันนั้น ๆ ในกรณีนี้ ตัวอย่างเพียงแค่กำหนดโครงสร้างของฟังก์ชันไว้ก่อน โดยยังไม่ได้ลงรายละเอียดของการทำงานจริงของแต่ละฟังก์ชัน คำสั่ง `pass` จึงทำหน้าที่แทนการเขียนโค้ดที่ยังไม่เสร็จสมบูรณ์ ทำให้โปรแกรมสามารถทำงานได้โดยไม่เกิดข้อผิดพลาดในระหว่างที่โค้ดยังอยู่ในขั้นตอนการพัฒนา

การมองหาแบบแผนเป็นขั้นตอนที่ 2 ของการคิดเชิงคำนวณ

หากพิจารณาอย่างละเอียด จะพบว่าปัญหาย่อยที่ 1 และปัญหาย่อยที่ 3 มีรูปแบบแบบแผนปัญหาเช่นเดียวกัน และต้องการการแก้ปัญหาที่เหมือนกัน โดยทั้งสองปัญหาเกี่ยวข้องกับการซ่อมแซมหลุมที่อยู่ข้าง ๆ จุดที่คาเรลยืนอยู่ ดังนั้น จึงสามารถใช้ฟังก์ชันเดียวกันในการแก้ไขทั้งสองปัญหาได้ (แต่ต้องให้คาเรลยืนอยู่ที่จุดฝั่งซ้ายของหลุมก่อนเรียกฟังก์ชัน) ซึ่งจะช่วยลดความซ้ำซ้อนของโค้ดรวมถึงทำให้โปรแกรมมีประสิทธิภาพมากขึ้นได้



ภาพที่ 1.7 กรอบสีแดงแสดงให้ว่าสองปัญหาย่อยมีลักษณะแบบแผนของปัญหาที่เหมือนกัน

เพราะฉะนั้นในการแก้ปัญหานี้จะมีการเรียกใช้ฟังก์ชันซ้ำในโค้ด เพื่อแก้ปัญหาย่อยที่มีแบบแผนเดียวกัน ฟังก์ชัน `fill_hole1()` และ `fill_hole2()` จึงถูกแทนที่ด้วยการเรียกฟังก์ชัน `fill_hole()` สองครั้งในตำแหน่งที่เหมาะสมดังนี้

```
def main():
 fill_hole()
 move_to_next_hole()
 fill_hole()

def fill_hole():
    pass

def move_to_next_hole()
    move()
    move()
    move()
```

ฟังก์ชัน `main` ที่ออกแบบใหม่นี้กระชับและเข้าใจง่ายกว่าฟังก์ชันเดิมอย่างมาก เนื่องจากฟังก์ชันใหม่นี้ได้มีการแบ่งปัญหาออกเป็นส่วนย่อย และนำการสังเกตแบบแผนของปัญหามาช่วยในการลดปริมาณโค้ดที่ต้องเขียนลง ส่งผลให้โค้ดที่แก้ไขนี้เรียบง่ายและลดโอกาสการเกิดข้อผิดพลาดลงได้

ยิ่งประหยัดการเขียนโค้ดลงแค่ไหน ยิ่งลดโอกาสที่จะเขียนโปรแกรมที่ผิดพลาดลงได้มากขึ้นเท่านั้น

สำหรับฟังก์ชันที่ต้องเขียนเพิ่มเติมนั้น มีเพียงฟังก์ชัน `fill_hole()` ซึ่งถูกเรียกใช้สองครั้งในโปรแกรม ผู้เขียนโปรแกรมสามารถใช้หลักการนำโค้ดมาใช้ซ้ำ (code reuse) โดยเขียนฟังก์ชันนี้เพียงครั้งเดียวแล้วนำมาใช้อีกครั้ง นอกจากนี้ ผู้เขียนโปรแกรมยังสามารถนำแนวคิดการแบ่งปัญหาออกเป็นส่วนย่อยมาใช้กับฟังก์ชัน `fill_hole()` เพื่อแยกการทำงานออกเป็นขั้นตอนย่อย ๆ ลงไปอีกได้ ซึ่งจะทำให้โค้ดมีความชัดเจน เป็นระบบ และจัดการได้ง่ายยิ่งขึ้น โดยสามารถแบ่งขั้นตอนย่อยลงไปได้อีก ดังนี้

1. กระโดดลงไปในหลุม
2. วางกระดิ่ง
3. กระโดดออกมานอกหลุม

หลังจากนั้นผู้เขียนโปรแกรมสามารถเริ่มลงมือเขียนโค้ด (อิมพลิเมนต์) อัลกอริทึมในการ `fill_hole()` ได้ดังนี้

อิมพลิเมนต์ (implement) หมายถึงการลงมือเขียนโค้ดจริง ตามที่ได้คิดและวางแผนการแก้ปัญหาเรียบร้อยแล้ว ซึ่งสอดคล้องกับขั้นตอนที่ 4 ของกระบวนการคิดเชิงคำนวณ

```
def fill_hole():
    jump_in()
    put_beeper()
    jump_out()

def jump_in():
    move()
    turn_right()
    move()

def jump_out():
    turn_around()
    move()
    turn_right()
    move()
```

หากสังเกตโดยละเอียด โค้ดข้างบนนี้มีลักษณะแบบแผนอีกหนึ่งรูปแบบ คือ `jump_in()` กับ `jump_out()` มีส่วนของโค้ดที่เหมือนกันอยู่ถึงสามบรรทัด นั่นก็คือ `move()` `turn_right()` `move()` ซึ่งคือแบบแผนการสั่งให้คาเรลเดินเป็นรูปตัวแอล ดังนั้นเพื่อประหยัดพื้นที่และทำให้

การเรียกใช้ฟังก์ชันเป็นไปได้อย่างมีประสิทธิภาพมากขึ้น สามารถเขียนฟังก์ชัน `move_L()` มาใช้ซ้ำได้อีก ดังนี้

```
def fill_hole():
    move_L()
    put_beeper()
    jump_out()

def move_L():
    move()
    turn_right()
    move()

def jump_out():
    turn_around()
    move_L()
```

ในการแก้ปัญหาคาเรลซ่อมถนนข้อนี้ ผู้เขียนโปรแกรมได้ใช้หลักการคิดเชิงคำนวณ 3 ใน 4 ขั้นตอน ได้แก่ การแบ่งปัญหาออกเป็นส่วนย่อย การสังเกตแบบแผน และการออกแบบอัลกอริทึม อย่างไรก็ตาม ในโจทย์ข้อนี้ ไม่ได้มีการใช้การคิดแบบนามธรรมซึ่งเป็นขั้นตอนสำคัญอีกขั้นตอนหนึ่งของกระบวนการคิดเชิงคำนวณ ทั้งนี้ เมื่อผู้เขียนโปรแกรมต้องเผชิญกับปัญหาที่มีความซับซ้อนมากขึ้นในอนาคต จะเห็นได้ว่าการคิดแบบนามธรรมจะเข้ามามีบทบาทสำคัญอย่างยิ่งในการช่วยจัดการกับความซับซ้อนของปัญหา

วงวน `for`

โครงสร้างการควบคุม (control flow) เป็นลักษณะพิเศษของภาษาคอมพิวเตอร์ที่ควบคุมลำดับการรันโปรแกรม ลำดับว่าบรรทัดไหนเกิดก่อนหลัง และเกิดกี่ครั้ง

โปรแกรมที่ผู้เรียนได้เขียนมาจนถึงขณะนี้ เป็นโปรแกรมที่มีลักษณะการประมวลผลชุดคำสั่งตั้งแต่บรรทัดแรกไปจนถึงบรรทัดสุดท้าย โดยอาจมีการเรียกใช้ฟังก์ชันย่อยระหว่างการประมวลผล แต่ยังคงเป็นการทำงานตามลำดับจากบรรทัดแรกไปจนถึงบรรทัดสุดท้าย หรือเรียกได้ว่าประมวลผลทั้งหมด แต่ทั้งนี้ ภาษาคอมพิวเตอร์มีสิ่งที่เรียกว่า โครงสร้างการควบคุม (control flow) ซึ่งเป็นชุดคำสั่งพิเศษที่ช่วยให้ผู้เขียนโปรแกรมสามารถกำหนดได้ว่าจะประมวลผลโค้ดส่วนใด หรือต้องการประมวลผลจำนวนกี่ครั้ง การใช้โครงสร้างการควบคุมนี้ทำให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมเพื่อแก้ปัญหาต่าง ๆ ได้อย่างหลากหลายและครอบคลุมมากยิ่งขึ้น

วงวน `for` เป็นหนึ่งในโครงสร้างการควบคุม ซึ่งควบคุมการทำงานของโปรแกรมให้วนซ้ำ ๆ ตามจำนวนครั้งที่กำหนดไว้ล่วงหน้า ตัวอย่างเช่น ถ้าผู้เขียนโปรแกรมต้องการให้คาเรลวางกระดิ่งลงบนถนนที่มีหลุมบ่อ 5 หลุม โดยที่คาเรลต้องวางกระดิ่งลงในทุก ๆ หลุม โดยเริ่มจากหลุมที่ 1 จนถึงหลุมที่ 5 โค้ดที่เขียนด้วยวงวน `for` จะมีลักษณะดังนี้

```
def main():
    move()
    for i in range(5):
        put_beeper()
        move()
        turn_around
```

ซึ่งหมายความว่า โปรแกรมจะทำการประมวลผลกลุ่มคำสั่ง (code block) ที่อยู่ภายใต้คำสั่ง for ทั้งหมด 5 ครั้ง (นักพัฒนาโปรแกรมมักเรียกกระบวนการนี้อย่างไม่เป็นทางการว่า *ฟอร์ (for) ลูปไป 5 รอบ*) หรือเท่ากับว่าตัวโปรแกรมจะทำการประมวลผลโค้ดดังนี้

```
def main():
    move()
    put_beeper()
    move()
    put_beeper()
    move()
    put_beeper()
    move()
    put_beeper()
    move()
    put_beeper()
    move()
    turn_around()
```

การใช้คำสั่งวงวน `for` ทำให้ผู้เขียนโค้ดสามารถประมวลผลคำสั่งซ้ำ ๆ ได้โดยไม่จำเป็นต้องเขียนโค้ดซ้ำตามจำนวนรอบที่ต้องการประมวลผล ทั้งนี้ในการใช้คำสั่งวงวน `for` จำเป็นต้องระบุจำนวนรอบที่ต้องการให้โปรแกรมทำงานล่วงหน้าโดยการกำหนดจำนวนรอบไว้ในโค้ด เมื่อโปรแกรมทำการประมวลผลคำสั่งการวนซ้ำจนถึงจำนวนรอบที่กำหนดแล้ว โปรแกรมจะประมวลผลคำสั่งถัดไปที่อยู่นอกโครงสร้างการวนซ้ำ (ซึ่งนักพัฒนาโปรแกรมมักเรียกอย่างไม่เป็นทางการว่า *ออกจากลูป*) โดยสรุป คำสั่งวงวน `for` มีโครงสร้างหรือรูปแบบการใช้งานดังนี้

```
for i in range(จำนวนครั้งที่จะวนซ้ำ):
  คำสั่งที่ต้องการวนซ้ำ
  คำสั่งที่ต้องการวนซ้ำ
```

กลุ่มคำสั่ง (code block) ที่ต้องการให้ทำงานซ้ำภายใต้คำสั่ง `for` จำเป็นต้องมีการจัดย่อหน้า (indentation) ให้เห็นอย่างชัดเจนว่าคำสั่งใดบ้างที่อยู่ภายในโครงสร้างการวนซ้ำ `for` การย่อหน้าเช่นนี้เหมือนกับการกำหนดโครงสร้างภายในฟังก์ชัน ซึ่งช่วยให้โปรแกรมสามารถแยกแยะได้ว่าคำสั่งใดที่เป็นส่วนหนึ่งของการวนซ้ำ และคำสั่งใดที่เป็นส่วนหนึ่งของโปรแกรมที่อยู่นอกการวนซ้ำ

สำหรับ i ใน range (number) นั้นเป็นเสมือนตัวแปรที่ใช้นับรอบของลูป ในการประมวลผลแต่ละครั้ง i ก็จะมีค่าที่เปลี่ยนไปในแต่ละรอบ เพื่อเป็นตัวแปรที่บอกว่าการวนซ้ำนี้เป็นครั้งที่เท่าไหร่ หรือลำดับใด เช่น หาก for i in range(5) i ก็จะมีค่าตั้งแต่ 0-4 ในแต่ละรอบ

วงวน `while`

โครงสร้างการควบคุมอีกประเภทหนึ่งที่พบได้บ่อยในการเขียนโปรแกรมคือการใช้คำสั่ง `while` ซึ่งเป็นการวนซ้ำคำสั่งเช่นเดียวกันกับคำสั่ง `for` โดยลักษณะสำคัญของวงวน `while` คือผู้เขียนโปรแกรมต้องกำหนดเงื่อนไขให้โปรแกรมวนซ้ำตามที่กำหนด เมื่อเงื่อนไขที่ระบุไม่เป็นจริง โปรแกรมจะหยุดการทำงานของโครงสร้างการวนซ้ำนี้ กล่าวคือคำสั่งวงวน while ทำงานโดยการตรวจสอบเงื่อนไขก่อนเสมอว่าเป็นจริงหรือไม่ หากเงื่อนไขเป็นจริง โปรแกรมจะดำเนินการคำสั่งภายในโครงสร้างการวนซ้ำต่อไป และจะหยุดการวนซ้ำ หรือที่เรียกกันว่าหลุดออกจากโครงสร้างเมื่อเงื่อนไขไม่เป็นจริง