คอมพิวเตอร์โปรแกรมมิ่ง

เริ่มต้นเขียนโปรแกรมด้วยภาษาไพธอน

อนิราช มิ่งขวัญ

ตำราประกอบการเรียนวิชา
060233115 - การเขียนโปรแกรมคอมพิวเตอร์

หลักสูตรวิศวกรรมสารสนเทศและเครือข่าย ภาควิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีและการจัดการอุตสาหกรรม มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ
ปราจีนบุรี

18 กุมภาพันธ์ 2025

คำนำ

หนังสือเล่มนี้มีจุดมุ่งหมายเพื่อเป็นแนวทางในการเรียนรู้และพัฒนาทักษะด้านการเขียนโปรแกรมคอมพิวเตอร์ด้วยภาษา Python ซึ่งเป็นหนึ่งในภาษาที่ได้รับความนิยมสูงในปัจจุบัน เนื้อหาในเล่มครอบคลุมทั้งแนวคิดพื้นฐานและการประยุกต์ใช้จริง ตั้งแต่โครงสร้างภาษา ตัวแปร ฟังก์ชัน การควบคุมการทำงานของโปรแกรม ไปจนถึงการทำงานกับไฟล์ โมดูล ไลบรารี และการจัดการข้อผิดพลาด

หนังสือเล่มนี้ยังเน้นการปูพื้นฐานเชิงตรรกะการเขียนโปรแกรมอย่างเป็นระบบ และเสริมสร้างความเข้าใจในแนวคิดเชิงวัตถุ (Object-Oriented Programming) ที่จำเป็นสำหรับการพัฒนาระบบขนาดใหญ่ในอนาคต นอกจากนี้ยังมีตัวอย่างการใช้งานจริง เช่น การเขียนโปรแกรมเพื่อจัดการข้อมูล การสร้างเกมอย่างง่าย และการวิเคราะห์ข้อมูลเบื้องต้น

ภาษา Python เป็นภาษาที่เหมาะสำหรับผู้เริ่มต้น เนื่องจากมีโครงสร้างที่อ่านง่าย ใช้งานสะดวก และสามารถนำไปใช้ได้หลากหลาย ตั้งแต่การพัฒนาเว็บไซต์ วิทยาการข้อมูล ไปจนถึงปัญญาประดิษฐ์ หนังสือเล่มนี้จึงเหมาะทั้งสำหรับผู้ที่เพิ่งเริ่มต้นเรียนรู้การเขียนโปรแกรม และผู้ที่มีพื้นฐานอยู่แล้วและต้องการเสริมสร้างความเข้าใจเชิงลึก

หวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะช่วยให้ผู้อ่านสามารถเขียนโปรแกรมได้อย่างมั่นใจ สร้างสรรค์โครงการต่าง ๆ ได้ด้วยตนเอง และวางรากฐานที่แข็งแกร่งสำหรับการพัฒนาทักษะด้านวิทยาการคอมพิวเตอร์และซอฟต์แวร์ในอนาคต

ขอขอบคุณผู้อ่านทุกท่านที่ให้ความสนใจ และขอให้ทุกท่านประสบความสำเร็จในการเดินทางสู่การเป็นนักพัฒนาโปรแกรมที่มีคุณภาพ

ด้วยความเคารพอย่างสูง

รศ.ดร. อนิราช มิ่งขวัญ
18 กุมภาพันธ์ 2025

กิตติกรรมประกาศ

หนังสือ Computer Programming with Python เล่มนี้เกิดขึ้นจากความมุ่งมั่นในการพัฒนาทรัพยากรการเรียนรู้ที่สามารถส่งเสริมทักษะการเขียนโปรแกรมของนิสิต นักศึกษา และผู้ที่สนใจด้านวิทยาการคอมพิวเตอร์ให้สามารถนำไปใช้งานได้จริง ทั้งในการเรียน การวิจัย และการประกอบวิชาชีพ

ผู้เขียนขอขอบพระคุณคณาจารย์ในสาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ที่ได้ให้คำแนะนำและข้อเสนอแนะอันมีค่าในระหว่างกระบวนการจัดทำหนังสือเล่มนี้ รวมถึงนิสิตนักศึกษาที่มีส่วนร่วมในการทดลองใช้ต้นฉบับ และให้ความคิดเห็นที่ช่วยพัฒนาเนื้อหาให้ดียิ่งขึ้น

ขอขอบคุณหน่วยงานและองค์กรต่าง ๆ ที่ให้การสนับสนุนด้านข้อมูล เครื่องมือ และทรัพยากรที่ใช้ในการเรียบเรียงหนังสือเล่มนี้ โดยเฉพาะชุมชนผู้ใช้ภาษา Python ที่ได้เผยแพร่ความรู้และแนวทางปฏิบัติที่เป็นประโยชน์อย่างยิ่ง

ท้ายที่สุดนี้ ผู้เขียนขอขอบคุณครอบครัวและผู้สนับสนุนทุกท่านที่อยู่เบื้องหลังความพยายามและการทำงานอย่างต่อเนื่อง ด้วยกำลังใจและความเข้าใจอย่างลึกซึ้ง

หากมีข้อผิดพลาดใด ๆ ในเนื้อหาของหนังสือเล่มนี้ ผู้เขียนขอรับไว้แต่เพียงผู้เดียว และยินดีรับคำแนะนำเพื่อการปรับปรุงในโอกาสต่อไป

ด้วยความเคารพอย่างสูง

รศ.ดร. อนิราช มิ่งขวัญ
18 กุมภาพันธ์ 2025

สารบัญ

1 บทนำสู่การเขียนโปรแกรมและภาษาไพธอน **1**

1.1 การเขียนโปรแกรมคืออะไร . . . 1

1.1.1 อัลกอริทึม (Algorithm) . . . 2

1.1.2 ซอร์สโค้ด (Source Code) . . . 3

1.1.3 การคอมไพล์และการแปลผล (Compilation/Interpretation) . . . 4

1.1.4 ภาษาการเขียนโปรแกรม (s) . . . 6

1.2 ภาษาไพธอนคืออะไร (What is Python?) . . . 9

1.3 การติดตั้งภาษาไพธอน (Installing Python) . . . 10

แบบฝึกหัด . . . 12

ภาคผนวก . . . 13

2 ตัวแปร ชนิดข้อมูล และการรับข้อมูลจากผู้ใช้ **15**

2.1 วงจรการพัฒนาโปรแกรม . . . 15

2.1.1 รหัสเทียม/อัลกอริทึม (Pseudocode/Algorithm) . . . 16

2.1.2 ผังงาน (Flow Chart) . . . 18

2.1.3 แผนภูมิ IPO (Input Process Output) . . . 19

2.1.4 เขียนโปรแกรมจากอัลกอริทึม (Program From Algorithm) . . . 19

2.2 ตัวแปร () . . . 20

2.2.1 การประกาศและกำหนดค่าตัวแปรในภาษา Python . . . 20

2.2.2 กฎการตั้งชื่อตัวแปร (Variable Naming Rules) . . . 20

2.3 ชนิดของข้อมูล () . . . 23

2.3.1 ชนิดข้อมูลที่ใช้บ่อยในภาษา Python . . . 23

2.3.2 การแปลงชนิดข้อมูล (Type Conversion) . . . 24

2.4 ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators) . . . 25

2.5 การรับข้อมูลจากผู้ใช้ (User Input) . . . 29

2.5.1 การรับข้อมูลจากผู้ใช้ด้วย input() . . . 29

2.5.2 การแปลงประเภทข้อมูลที่รับเข้ามา . . . 30

2.6 ตัวอย่างการประยุกต์ใช้งานจริง (Practical Example) . . . 30

2.6.1 ตัวอย่างที่ 1: โปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้า . . . 30

2.6.2 ตัวอย่างที่ 2: โปรแกรมคำนวณค่าดัชนีมวลกาย (BMI) . . . 31

2.6.3 ตัวอย่างที่ 3: โปรแกรมแปลงอุณหภูมิ . . . 32

แบบฝึกหัด . . . 34

ภาคผนวก . . . 35

3 เงื่อนไขและการควบคุมลำดับการทำงาน **39**

3.1 คำสั่งเงื่อนไข (Conditional Statements) . . . 39

3.1.1 คำสั่ง If . . . 41

3.1.2 คำสั่ง Elif . . . 42

3.1.3 คำสั่ง Else . . . 45
3.1.4 Nested If . . . 47
3.2 ตัวดำเนินการเปรียบเทียบ . . . 50
3.2.1 การเปรียบเทียบค่า . . . 51
3.2.2 การเปรียบเทียบข้อความ . . . 51
3.3 ตัวดำเนินการทางตรรกะ . . . 52
3.4 ตัวดำเนินการเปรียบเทียบอัตลักษณ์ (Identity Operators) . . . 54
3.5 ตัวดำเนินการตรวจสอบสมาชิก(Membership Operators) . . . 55
3.6 การรวมตัวดำเนินการเปรียบเทียบและตรรกะ(Combining Comparison and Logical Operators) 56
3.7 ตัวอย่างการใช้งานจริง . . . 57
3.7.1 โปรแกรมตรวจสอบสถานะการเข้าศึกษา . . . 57
3.7.2 โปรแกรมตรวจสอบผ่านหรือไม่ผ่าน . . . 58
แบบฝึกหัด . . . 59
ภาคผนวก . . . 60

4 การใช้ลูป For และ While **63**
4.1 บทนำสู่ลูป (s) . . . 63
4.2 ลูป For . . . 65
4.2.1 การวนซ้ำผ่านรายการ (List) . . . 66
4.2.2 การวนซ้ำผ่านสตริง . . . 67
4.2.3 การใช้ฟังก์ชัน Range . . . 69
4.3 ลูป While . . . 71
4.3.1 ลูป While และเงื่อนไข . . . 72
4.3.2 ลูปไม่มีที่สิ้นสุด (Infinite Loops) . . . 72
4.3.3 เซนทิเนล (Sentinel) . . . 72
4.3.4 ลูปตรวจสอบความถูกต้องของข้อมูลที่รับเข้า (Input Validation Loop) . . . 74
4.4 คำสั่งควบคุมลูป (Loop Control Statements) . . . 74
4.4.1 คำสั่ง Break . . . 74
4.4.2 คำสั่ง Continue . . . 75
4.4.3 คำสั่ง Pass . . . 75
4.5 การคำนวณผลรวมสะสม (Running Total) . . . 75
4.6 ลูปซ้อน (Nested Loop) . . . 76
4.7 ตัวอย่างการใช้งานจริง . . . 76
4.7.1 ลูปซ้อน (Nested Loops) . . . 77
4.7.2 การวนซ้ำผ่าน Dictionary . . . 77
4.7.3 ผลรวมของจำนวนเต็มธรรมชาติ n ตัวแรก . . . 77
4.7.4 การคำนวณแฟกทอเรียล (Factorial) . . . 78
แบบฝึกหัด . . . 79
ภาคผนวก . . . 80

5 ฟังก์ชันและแนวคิดการแบ่งโมดูล **83**
5.1 ฟังก์ชัน (s) . . . 83
5.1.1 การกำหนดฟังก์ชัน . . . 84
5.1.2 การกำหนดฟังก์ชันอย่างง่าย . . . 84
5.1.3 พารามิเตอร์ . . . 86
5.1.4 พารามิเตอร์หลายตัว . . . 86
5.1.5 พารามิเตอร์ที่มีค่าเริ่มต้น . . . 86
5.1.6 อาร์กิวเมนต์ที่มีความยาวไม่จำกัด . . . 87

5.1.7 ค่าที่ส่งกลับจากฟังก์ชัน . . . 88
5.2 ขอบเขตของฟังก์ชันและตัวแปร (Function Scope and Variables) . . . 93
5.2.1 ขอบเขตของฟังก์ชัน (Function Scope) . . . 93
5.2.2 ตัวแปรสากล (Global Variables) . . . 94
5.2.3 การเปลี่ยนค่าตัวแปรสากล (Modifying Global Variables) . . . 94
5.3 การจัดเก็บฟังก์ชันในโมดูล (Storing Functions in Modules) . . . 95
5.3.1 ความเข้าใจเกี่ยวกับโมดูล (Understanding Modules) . . . 96
5.3.2 การสร้างและใช้งานโมดูล (Creating and Using Modules) . . . 96
5.3.3 การนำเข้าโมดูล (Importing Modules) . . . 97
5.3.4 ข้อดีของการใช้โมดูล (Benefits Using Modules) . . . 98
5.4 ฟังก์ชันแบบเรียกซ้ำ (Recursive Functions) . . . 98
5.4.1 ความเข้าใจเกี่ยวกับการเรียกซ้ำ (Understanding Recursion) . . . 98
5.4.2 ตัวอย่าง: การคำนวณแฟกทอเรียล (Factorial) . . . 98
5.4.3 ข้อดีของฟังก์ชันแบบเรียกซ้ำ (Advantages) . . . 99
5.4.4 ข้อเสียของฟังก์ชันแบบเรียกซ้ำ (Disadvantages) . . . 99
5.4.5 Tail Recursion . . . 99
5.4.6 ตัวอย่าง: ลำดับฟีโบนัชชี (Fibonacci Sequence) . . . 100
5.4.7 สรุป . . . 100
5.5 ฟังก์ชันที่ใช้บ่อยใน Python (Python Common Functions) . . . 100
5.6 ตัวอย่างการใช้งานจริง . . . 103
5.6.1 การคำนวณพื้นที่วงกลม . . . 103
5.6.2 ตัวอย่าง: การตรวจสอบจำนวนเฉพาะ . . . 103
5.6.3 ตัวอย่าง: การแยกโค้ดเป็นโมดูล . . . 104
แบบฝึกหัด . . . 106
ภาคผนวก . . . 107

6 ลิสต์และการดำเนินการพื้นฐาน 111
6.1 บทนำสู่ลิสต์ . . . 111
6.1.1 การสร้างและการเข้าถึงลิสต์ . . . 112
6.1.2 การเข้าถึงด้วยดัชนีลบ . . . 112
6.1.3 การแก้ไขลิสต์ . . . 113
6.2 เมธอดของลิสต์ . . . 113
6.2.1 Append . . . 114
6.2.2 Insert . . . 114
6.2.3 Remove . . . 115
6.2.4 Pop . . . 115
6.2.5 Index . . . 116
6.2.6 Clear . . . 116
6.2.7 Sort . . . 116
6.2.8 Reverse . . . 117
6.3 การตัดช่วงข้อมูล (Slicing) . . . 117
6.3.1 การเข้าถึงบางส่วนของลิสต์ . . . 117
6.3.2 การตัดช่วงด้วยดัชนีลบ . . . 118
6.3.3 การเข้าถึงบางส่วนของสตริง . . . 118
6.4 ฟังก์ชันในตัวสำหรับการทำงานกับลิสต์ . . . 119
6.5 ลิสต์สองมิติ . . . 121
6.6 ทูเพิลในภาษา Python . . . 123
6.6.1 การสร้างและเข้าถึงทูเพิล . . . 123

6.6.2 การแพ็กและแยกแพ็กทูเพิล . . . 124
6.6.3 เมธอดที่ใช้บ่อยของทูเพิล . . . 125
6.7 ตัวอย่างการใช้งานจริง . . . 128
6.7.1 ตัวอย่าง: การหาค่ามากที่สุดในลิสต์ . . . 128
6.7.2 ตัวอย่าง: การลบค่าซ้ำในลิสต์ . . . 128
6.7.3 ตัวอย่าง: List Comprehension . . . 129
แบบฝึกหัด . . . 130
ภาคผนวก . . . 131

7 เซต ดิกชันนารี และการดำเนินการพื้นฐาน 133
7.1 แนะนำเซต (Set) . . . 133
7.1.1 การสร้างและเข้าถึงเซต . . . 134
7.1.2 การจัดการสมาชิกของเซต . . . 134
7.1.3 การดำเนินการของเซต . . . 135
7.1.4 การอัปเดตการดำเนินการของเซต . . . 136
7.1.5 การดำเนินการของเซต: ซับเซตและซูเปอร์เซต . . . 137
7.2 บทนำสู่ . . . 139
7.2.1 การสร้างและเข้าถึง Dictionary . . . 139
7.2.2 การเข้าถึง Dictionary ด้วยการวนลูป . . . 140
7.2.3 การปรับปรุง Dictionary . . . 141
7.2.4 เมท็อดของ Dictionary . . . 142
7.3 ตัวอย่างการใช้งานจริง . . . 143
7.3.1 ตัวอย่าง: การใช้ Tuple เป็นคีย์ใน Dictionary . . . 143
7.3.2 ตัวอย่าง: การนับจำนวนสมาชิกในลิสต์ด้วย Dictionary . . . 144
7.3.3 ตัวอย่าง: การลบค่าซ้ำในลิสต์ด้วย Set . . . 144
7.3.4 ตัวอย่าง: การแปลงลิสต์เป็น Dictionary . . . 145
7.3.5 ตัวอย่าง: พลังของการใช้ Set กับ List . . . 145
แบบฝึกหัด . . . 147
ภาคผนวก . . . 148

8 การจัดการไฟล์และการดำเนินการ I/O 151
8.1 การจัดการไฟล์ . . . 151
8.1.1 การเปิดไฟล์ . . . 152
8.1.2 การปิดไฟล์ . . . 153
8.1.3 การใช้ with statement . . . 153
8.2 การเขียนไฟล์ . . . 153
8.2.1 การเขียนลงในไฟล์ . . . 154
8.2.2 การเพิ่มข้อมูลลงในไฟล์ . . . 154
8.3 การอ่านไฟล์ . . . 154
8.3.1 การอ่านไฟล์ทั้งหมด . . . 154
8.3.2 การอ่านแต่ละบรรทัด . . . 155
8.3.3 ตัวอย่างการประมวลผลไฟล์ข้อความ . . . 155
8.4 การทำงานกับเรคอร์ดและไฟล์ไบนารี . . . 157
8.4.1 การเขียนเรคอร์ดลงในไฟล์ไบนารี . . . 157
8.4.2 การอ่านเรคอร์ดจากไฟล์ไบนารี . . . 158
8.4.3 การทำงานกับหลายเรคอร์ด . . . 159
8.5 การใช้โหมด + ใน Python . . . 160
8.5.1 ตัวอย่างการใช้โหมด `r+` . . . 160

8.5.2 ตัวอย่างการใช้โหมด `w+` . . . 161
8.5.3 ตัวอย่างการใช้โหมด `a+` . . . 162
8.5.4 ข้อควรพิจารณาในการใช้งานไฟล์ไบนารี . . . 163
8.6 ตัวอย่างการใช้งานจริง . . . 163
8.6.1 ตัวอย่าง: คัดลอกเนื้อหาจากไฟล์หนึ่งไปยังอีกไฟล์ . . . 163
8.6.2 ตัวอย่าง: นับจำนวนคำในไฟล์ . . . 163
8.6.3 ตัวอย่าง: เขียนข้อมูลจากผู้ใช้ลงในไฟล์ . . . 164
แบบฝึกหัด . . . 165
ภาคผนวก . . . 166

9 การจัดการข้อผิดพลาด 169
9.1 บทนำสู่ข้อผิดพลาด . . . 170
9.2 ประเภทของข้อผิดพลาดที่พบบ่อย . . . 170
9.3 บล็อก Try-Except . . . 171
9.4 การจับข้อผิดพลาดหลายประเภท . . . 173
9.5 การจับข้อผิดพลาดทั้งหมด . . . 173
9.6 Else Block . . . 174
9.7 Finally Block . . . 175
9.8 Combining Try-Except-Else-Finally . . . 175
9.9 Defining and Using a Custom Exception in Python . . . 176
9.10 Practical Examples . . . 177
9.10.1 Example: Handling File Operations . . . 177
9.10.2 Example: Input Validation . . . 177
แบบฝึกหัด . . . 179
ภาคผนวก . . . 180

10 ไลบรารีและโมดูลของภาษาไพธอน 185
10.1 บทนำสู่โมดูลและไลบรารี . . . 185
10.1.1 โมดูลคืออะไร? . . . 185
10.1.2 ไลบรารีคืออะไร? . . . 186
10.2 การนำเข้าโมดูล . . . 186
10.2.1 การนำเข้าโมดูลทั้งหมด . . . 187
10.2.2 การนำเข้าส่วนเฉพาะจากโมดูล . . . 187
10.2.3 การนำเข้าโมดูลด้วยชื่อย่อ . . . 187
10.2.4 การนำเข้าทุกองค์ประกอบจากโมดูล . . . 187
10.3 ภาพรวมของไลบรารีมาตรฐานของ Python . . . 188
10.4 โมดูลที่ใช้งานบ่อยในไลบรารีมาตรฐาน . . . 188
10.4.1 โมดูล os . . . 188
10.4.2 โมดูล sys . . . 189
10.4.3 โมดูล datetime . . . 189
10.4.4 โมดูล random . . . 189
10.4.5 โมดูล json . . . 190
10.5 ภาพรวมของไลบรารีจากบุคคลที่สาม . . . 190
10.6 การติดตั้งไลบรารีจากบุคคลที่สาม . . . 190
10.7 ตัวอย่างไลบรารียอดนิยมจากบุคคลที่สาม . . . 190
10.7.1 NumPy . . . 191
10.7.2 Pandas . . . 191

10.7.3 Matplotlib . . . 192
10.7.4 Requests . . . 194
10.7.5 Flask . . . 195
10.7.6 SQLite . . . 196
10.8 ตัวอย่างการใช้งานจริง . . . 197
10.8.1 ตัวอย่าง: การใช้โมดูลหลายตัวร่วมกัน . . . 198
แบบฝึกหัด . . . 199
ภาคผนวก . . . 200

11 การเขียนโปรแกรมเชิงวัตถุ **205**
11.1 บทนำสู่การเขียนโปรแกรมเชิงวัตถุ . . . 205
11.2 คลาสและอ็อบเจกต์ . . . 207
11.3 การสืบทอด . . . 210
11.4 การห่อหุ้ม . . . 211
11.5 พอลิมอร์ฟิซึม . . . 214
11.6 ตัวอย่างการใช้งานจริง . . . 217
11.6.1 ตัวอย่าง: การสร้างคลาสบัญชีธนาคารแบบง่าย . . . 218
11.6.2 ตัวอย่าง: การสืบทอดและการเขียนเมธอดซ้ำ . . . 218
11.7 การสาธิตระบบห้องสมุดที่ใช้ OOP . . . 219
แบบฝึกหัด . . . 224
ภาคผนวก . . . 225

12 การเขียนโปรแกรมเชิงฟังก์ชัน **229**
12.1 บทนำสู่การเขียนโปรแกรมเชิงฟังก์ชัน . . . 229
12.2 แนวคิดหลักของการเขียนโปรแกรมเชิงฟังก์ชัน . . . 230
12.2.1 ฟังก์ชันบริสุทธิ์(Pure functions) . . . 230
12.2.2 ฟังก์ชันลำดับสูง(Higher-order functions) . . . 230
12.2.3 ความไม่เปลี่ยนแปลงของข้อมูล(Immutability) . . . 231
12.2.4 ฟังก์ชันเป็นพลเมืองชั้นหนึ่ง(First-class functions) . . . 231
12.3 ฟังก์ชันแลมบ์ดา (Lambda Functions) . . . 231
12.4 Map . . . 232
12.5 Filter . . . 232
12.6 Reduce . . . 232
12.7 การเวียนกลับ (Recursion) . . . 234
แบบฝึกหัด . . . 235
ภาคผนวก . . . 236

บรรณานุกรม **239**

บทที่ 1

บทนำสู่การเขียนโปรแกรมและภาษาไพธอน

บทนี้นำเสนอภาพรวมของการเขียนโปรแกรมและภาษาไพธอนอย่างครอบคลุม โดยครอบคลุมแนวคิดพื้นฐาน ประวัติความเป็นมา การประยุกต์ใช้งาน และขั้นตอนเชิงปฏิบัติในการติดตั้งภาษาไพธอนและเขียนโปรแกรมแรกของคุณ การเข้าใจแนวคิดพื้นฐาน เช่น อัลกอริทึม ซอร์สโค้ด และความแตกต่างระหว่างการคอมไพล์และการแปลผล จะช่วยให้ผู้เรียนมีพื้นฐานที่มั่นคงในการทำความเข้าใจหลักการทำงานของโปรแกรมคอมพิวเตอร์ บทนี้เน้นย้ำถึงความสำคัญของทักษะการเขียนโปรแกรมในการเสริมสร้างกระบวนการคิดอย่างเป็นระบบ การแก้ปัญหา การเปิดโอกาสในสายอาชีพที่หลากหลาย และการทำงานอัตโนมัติสำหรับงานที่ซ้ำซาก นอกจากนี้ ยังกล่าวถึงประเภทของภาษาการเขียนโปรแกรม โดยแยกแยะระหว่างภาษาในระดับต่ำซึ่งใกล้เคียงกับภาษาเครื่อง กับภาษาระดับสูง เช่น ไพธอน ซึ่งอ่านเข้าใจง่ายกว่าและเรียนรู้ได้ง่ายกว่า

นอกจากนี้ บทนี้ยังเจาะลึกเกี่ยวกับภาษาไพธอนโดยเฉพาะ โดยอธิบายคุณลักษณะเด่น เช่น ความสามารถในการอ่านเข้าใจง่าย ไลบรารีที่หลากหลาย ความยืดหยุ่นในการใช้งาน และการสนับสนุนจากชุมชนผู้ใช้ที่แข็งแกร่ง โดยกล่าวถึงประวัติของไพธอนตั้งแต่การพัฒนาโดย *Guido van Rossum ในปี ค.ศ. 1991* จนกลายเป็นหนึ่งในภาษาการเขียนโปรแกรมที่ได้รับความนิยมมากที่สุดในโลก ส่วนที่เป็นภาคปฏิบัติจะนำผู้อ่านไปสู่ขั้นตอนการดาวน์โหลดและติดตั้งไพธอน การตั้งค่าสภาพแวดล้อมการพัฒนา และการเขียนและรันโปรแกรมไพธอนโปรแกรมแรก เมื่อสิ้นสุดบทนี้ ผู้อ่านจะมีความรู้และทักษะที่จำเป็นในการเริ่มต้นเส้นทางของตนในการเขียนโปรแกรมด้วยภาษาไพธอน พร้อมที่จะสำรวจการประยุกต์ใช้งานในด้านการพัฒนาเว็บ วิทยาการข้อมูล การเรียนรู้ของเครื่อง และระบบอัตโนมัติ

1.1 การเขียนโปรแกรมคืออะไร

การเขียนโปรแกรมคือการออกแบบและสร้างโปรแกรมคอมพิวเตอร์ที่สามารถทำงานได้จริง เพื่อตอบสนองต่อภารกิจเฉพาะด้านคอมพิวเตอร์ โดยจะเป็นการเขียนโค้ดด้วยภาษาการเขียนโปรแกรมที่คอมพิวเตอร์สามารถเข้าใจและประมวลผลได้ กระบวนการนี้มีความสำคัญอย่างยิ่งในยุคดิจิทัลปัจจุบัน เพราะช่วยให้มนุษย์สามารถควบคุมการทำงานระหว่างมนุษย์และเครื่องจักร ทำงานอัตโนมัติ และแก้ปัญหาที่ซับซ้อนอย่างมีประสิทธิภาพและแม่นยำ หลักการของการเขียนโปรแกรมคอมพิวเตอร์กำลังถูกนำไปประยุกต์ใช้กับเทคโนโลยีเกิดใหม่ เช่น ปัญญาประดิษฐ์ การรู้จำเสียงพูด และอินเทอร์เน็ตของสรรพสิ่ง (IoT) ซึ่งทำให้การเขียนโปรแกรมกลายเป็นทักษะพื้นฐานสำหรับทั้งมืออาชีพและบุคคลทั่วไป ผู้ที่มีทักษะการเขียนโปรแกรมสามารถสร้างสรรค์ไอเดีย พัฒนาแอปพลิเคชันซอฟต์แวร์ และคิดค้นทางแก้ปัญหาใหม่ ๆ ที่สามารถตอบโจทย์ความท้าทายในชีวิตจริง นอกจากนี้ การเขียนโปรแกรมยังส่งเสริมการคิดเชิงคำนวณ การแก้ปัญหา และความคิดสร้างสรรค์ จึงเป็นเครื่องมือสำคัญสำหรับนวัตกรรมและผู้ประกอบการยุคใหม่

แนวคิดสำคัญ:

- **อัลกอริทึม (Algorithm):** ขั้นตอนการแก้ปัญหาอย่างเป็นลำดับ
- **ซอร์สโค้ด (Source Code):** คำสั่งที่มนุษย์เขียนและสามารถอ่านได้
- **การคอมไพล์/การแปลผล (Compilation/Interpretation):** กระบวนการแปลงซอร์สโค้ดให้กลายเป็นรหัสเครื่องที่คอมพิวเตอร์สามารถประมวลผลได้
- **ภาษาการเขียนโปรแกรม (Programming Languages):** ภาษาที่ใช้ในการเขียนโปรแกรม เช่น Python, Java, C++

1.1.1 อัลกอริทึม (Algorithm)

อัลกอริทึม () [1] คือขั้นตอนหรือชุดคำสั่งที่มีการจัดเรียงอย่างชัดเจน เพื่อทำงานใดงานหนึ่งหรือแก้ไขปัญหาที่เจาะจง เปรียบได้กับสูตรอาหารในตำรา ที่ระบุขั้นตอนแต่ละขั้นเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ เช่นเดียวกับสูตรอาหารที่มีวิธีประกอบอาหารอย่างละเอียด อัลกอริทึมก็มีขั้นตอนที่ชัดเจนเพื่อบรรลุเป้าหมายของโปรแกรม

ตัวอย่างง่าย ๆ เพื่อช่วยให้เข้าใจ:

ตัวอย่าง: อัลกอริทึมในการทำแซนด์วิช

1. **เตรียมวัตถุดิบและอุปกรณ์:** ขนมปัง เนย ชีส มีด และจาน
2. **ทาเนย:** ใช้มีดทาเนยลงบนด้านหนึ่งของขนมปังแต่ละแผ่น
3. **วางชีส:** วางชีสลงบนด้านที่ทาเนยของแผ่นขนมปังแผ่นหนึ่ง
4. **ประกบขนมปัง:** นำขนมปังอีกแผ่นประกบด้านบน โดยให้ด้านที่ทาเนยหันเข้าด้านใน
5. **เสิร์ฟ:** วางแซนด์วิชลงบนจานและพร้อมเสิร์ฟ

ในบริบทของการเขียนโปรแกรม อัลกอริทึมก็มีแนวทางคล้ายกัน:

ตัวอย่าง: อัลกอริทึมในการบวกเลขสองจำนวน

1. **เริ่มต้น:** เริ่มกระบวนการ
2. **รับข้อมูลเข้า:** รับตัวเลขสองจำนวนจากผู้ใช้
3. **ประมวลผล:** ทำการบวกตัวเลขทั้งสอง
4. **แสดงผลลัพธ์:** แสดงผลรวมที่ได้
5. **สิ้นสุด:** จบกระบวนการ

ตัวอย่างต่อไปนี้แสดงให้เห็นว่าอัลกอริทึมสามารถแปลงเป็นโปรแกรมไพธอนได้อย่างไร: ตัวอย่างนี้แสดงให้เห็นว่าอัลกอริธึมสามารถแปลงเป็นโปรแกรมไพธอนแบบง่าย ๆ ได้อย่างไร โดยโปรแกรมจะรับข้อมูลจากผู้ใช้ ประมวลผล และแสดงผลลัพธ์ที่ต้องการ อัลกอริธึมเป็นพื้นฐานสำคัญของการเขียนโปรแกรม เพราะช่วยกำหนดแนวทางในการแก้ปัญหาได้อย่างชัดเจน และสามารถนำไปใช้กับงานที่หลากหลาย ตั้งแต่การคำนวณพื้นฐานไปจนถึงการวิเคราะห์ข้อมูลที่ซับซ้อน

```python
#Add2Numbers.py
# Step 1: Start
# Step 2: Input
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# Step 3: Process
sum = number1 + number2

# Step 4: Output
print("The sum is:", sum)

# Step 5: End
```

Listing 1.1: Example: Algorithm to Add Two Numbers

1.1.2 ซอร์สโค้ด (Source Code)

ซอร์สโค้ด() หมายถึงชุดคำสั่งและข้อความที่เขียนโดยโปรแกรมเมอร์โดยใช้ภาษาการเขียนโปรแกรม คำสั่งเหล่านี้สามารถอ่านเข้าใจได้โดยมนุษย์ ซึ่งแตกต่างจากรหัสเครื่องที่อยู่ในรูปแบบเลขฐานสองซึ่งสามารถอ่านได้เฉพาะโดยคอมพิวเตอร์เท่านั้น

ประเด็นสำคัญเกี่ยวกับซอร์สโค้ด:

1. **อ่านได้โดยมนุษย์:** ซอร์สโค้ดเขียนด้วยภาษาการเขียนโปรแกรมที่มีลักษณะคล้ายภาษามนุษย์ เช่น Python, Java และ C++ ซึ่งเป็นภาษาที่นิยมใช้ในการเขียนซอร์สโค้ด

2. **คำสั่ง:** โค้ดประกอบด้วยชุดคำสั่งที่บอกให้คอมพิวเตอร์ดำเนินการตามที่กำหนด ตั้งแต่คำสั่งง่าย ๆ เช่น การบวกเลข ไปจนถึงกระบวนการซับซ้อน เช่น การรันเว็บเซิร์ฟเวอร์

3. **การแก้ไขและปรับปรุง:** โปรแกรมเมอร์สามารถแก้ไขและปรับปรุงซอร์สโค้ดได้อย่างง่ายดาย ทำให้สามารถพัฒนาและดูแลระบบซอฟต์แวร์อย่างต่อเนื่อง

```python
#Add2Numbers.py
# This is a comment explaining that the following lines of code will add two numbers

# Input: Asking the user to enter two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# Process: Adding the two numbers
sum = number1 + number2

# Output: Displaying the result
print("The sum is:", sum)
```

Listing 1.2: Example of Source Code in Python

คำอธิบาย:

- **คำอธิบาย (Comments):** บรรทัดที่ขึ้นต้นด้วยเครื่องหมาย # เป็นคำอธิบาย ซึ่งคอมพิวเตอร์จะไม่ประมวลผล แต่ช่วยให้โปรแกรมเมอร์เข้าใจโค้ดได้ง่ายขึ้น
- **Input: input("Enter the first number: ")** เป็นคำสั่งให้ผู้ใช้ป้อนตัวเลข และ **int()** ใช้แปลงค่าที่รับเข้ามาให้เป็นจำนวนเต็ม
- **ตัวแปร: number1 และ number2** เป็นตัวแปรที่ใช้เก็บค่าตัวเลขที่ผู้ใช้ป้อนเข้ามา
- **กระบวนการ: sum = number1 + number2** เป็นขั้นตอนการคำนวณผลรวมของสองตัวเลข
- **การแสดงผล: print("The sum is:", sum)** ใช้แสดงผลลัพธ์ให้ผู้ใช้เห็น

เหตุผลที่ซอร์สโค้ดมีความสำคัญ:

- **การสื่อสาร:** ช่วยให้โปรแกรมเมอร์สามารถสื่อสารคำสั่งไปยังคอมพิวเตอร์ได้ในรูปแบบที่เป็นระบบและเข้าใจง่าย
- **การทำงานร่วมกัน:** โปรแกรมเมอร์หลายคนสามารถทำงานร่วมกันในโปรเจกต์เดียวกันได้โดยการอ่านและแก้ไขซอร์สโค้ด
- **การตรวจหาข้อผิดพลาด:** เมื่อตรวจพบข้อผิดพลาด โปรแกรมเมอร์สามารถอ่านซอร์สโค้ดเพื่อหาสาเหตุและแก้ไขได้
- **การบำรุงรักษา:** ซอร์สโค้ดสามารถปรับปรุงเพิ่มเติมเพื่อเพิ่มคุณสมบัติใหม่ หรือเพิ่มประสิทธิภาพในการทำงานได้ในอนาคต

กล่าวโดยสรุป ซอร์สโค้ดคือแบบแปลน (blueprint) ของซอฟต์แวร์ทุกประเภท เป็นพื้นที่ที่โปรแกรมเมอร์นำตรรกะ ความคิดสร้างสรรค์ และทักษะในการแก้ปัญหามาใช้เพื่อสร้างโปรแกรมที่ทำงานได้จริงและมีประสิทธิภาพ

1.1.3 การคอมไพล์และการแปลผล (Compilation/Interpretation)

การคอมไพล์และการแปลผลเป็นกระบวนการที่ใช้ในการแปลงซอร์สโค้ด ซึ่งเขียนด้วยภาษาการเขียนโปรแกรมที่มนุษย์สามารถอ่านได้ ให้กลายเป็นรหัสเครื่อง (machine code) ซึ่งเป็นรหัสเลขฐานสองที่หน่วยประมวลผลของคอมพิวเตอร์สามารถดำเนินการได้โดยตรง

การคอมไพล์ () การคอมไพล์คือการแปลซอร์สโค้ดทั้งหมดของโปรแกรมให้เป็นรหัสเครื่องก่อนที่โปรแกรมจะถูกรัน กระบวนการนี้จะดำเนินการโดยคอมไพเลอร์ (compiler) เมื่อซอร์สโค้ดได้รับการคอมไพล์แล้ว จะได้ไฟล์ปฏิบัติการ (executable file) ที่สามารถนำไปรันได้โดยไม่ต้องมีซอร์สโค้ดต้นฉบับ

ขั้นตอนของการคอมไพล์:

1. **เขียนซอร์สโค้ด:** โปรแกรมเมอร์เขียนซอร์สโค้ดด้วยภาษาระดับสูง (เช่น C, C++)

2. **คอมไพล์:** คอมไพเลอร์แปลซอร์สโค้ดทั้งหมดเป็นรหัสเครื่องและสร้างไฟล์ปฏิบัติการ

3. **รันโปรแกรม:** ระบบปฏิบัติการของคอมพิวเตอร์รันไฟล์ปฏิบัติการนั้น

```c
//Add2Numbers.c
// C program to add two numbers
#include <stdio.h>

int main() {
    int number1, number2, sum;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    sum = number1 + number2;

    printf("Sum: %d\n", sum);
    return 0;
}
```

Listing 1.3: Example of Source Code in C

ในตัวอย่างนี้ โค้ดภาษา C สำหรับการบวกเลขสองจำนวนจะถูกคอมไพล์เป็นไฟล์ปฏิบัติการก่อนจึงจะสามารถรันได้

การแปลผล () ในทางตรงกันข้าม การแปลผลคือการแปลซอร์สโค้ดเป็นรหัสเครื่องทีละบรรทัดในขณะรันโปรแกรม โดยกระบวนการนี้จะทำโดยตัวแปล (interpreter) โปรแกรมที่ถูกแปลผลจะไม่สร้างไฟล์ปฏิบัติการ แต่ตัวแปลจะอ่านและดำเนินการซอร์สโค้ดโดยตรง

ขั้นตอนของการแปลผล:

1. **เขียนซอร์สโค้ด:** โปรแกรมเมอร์เขียนซอร์สโค้ดด้วยภาษาระดับสูง (เช่น Python, JavaScript)

2. **แปลผล:** ตัวแปลจะอ่านและดำเนินการซอร์สโค้ดทีละบรรทัด

3. **รันโปรแกรม:** โปรแกรมถูกรันโดยตรงจากซอร์สโค้ด

```python
#Add2Numbers.py
# Python program to add two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

sum = number1 + number2

print("Sum:", sum)
```

Listing 1.4: Example of Source Code in Python

ในตัวอย่างนี้ โค้ดภาษา Python จะถูกแปลผลและดำเนินการทีละบรรทัดในขณะรัน

ความแตกต่างที่สำคัญ:

- **ความเร็ว:** โปรแกรมที่ถูกคอมไพล์มักจะทำงานได้เร็วกว่า เพราะซอร์สโค้ดทั้งหมดถูกแปลงเป็นรหัสเครื่องล่วงหน้า ในขณะที่โปรแกรมที่แปลผลจะทำงานช้ากว่า เพราะแปลงโค้ดในขณะรัน
- **การตรวจจับข้อผิดพลาด:** คอมไพเลอร์สามารถตรวจพบข้อผิดพลาดได้ก่อนโปรแกรมถูกรัน ขณะที่ตัวแปลจะตรวจพบข้อผิดพลาดเมื่อโปรแกรมทำงาน
- **ความยืดหยุ่น:** ตัวแปลให้ความยืดหยุ่นในการทดสอบและดีบักมากกว่า เพราะสามารถรันโค้ดได้ทันทีโดยไม่ต้องคอมไพล์ใหม่

เหตุผลที่กระบวนการเหล่านี้มีความสำคัญ:

- **ประสิทธิภาพ:** การเข้าใจกระบวนการเหล่านี้ช่วยให้เลือกภาษาการเขียนโปรแกรมและเครื่องมือที่เหมาะสมกับงาน
- **การจัดการข้อผิดพลาด:** การทราบว่าภาษาใดเป็นภาษาแบบคอมไพล์หรือแปลผลช่วยให้คาดการณ์ได้ว่าเมื่อใดข้อผิดพลาดจะถูกตรวจพบ
- **สมรรถนะของโปรแกรม:** ภาษาที่ถูกคอมไพล์มักเหมาะสำหรับโปรแกรมที่ต้องการประสิทธิภาพสูง ส่วนภาษาที่แปลผลจะเหมาะกับการพัฒนาอย่างรวดเร็วและการทดลองแนวคิดใหม่ ๆ

โดยสรุป การคอมไพล์และการแปลผลเป็นกระบวนการสำคัญในการแปลงซอร์สโค้ดที่มนุษย์เขียนให้อยู่ในรูปแบบที่คอมพิวเตอร์สามารถดำเนินการได้ โดยแต่ละกระบวนการมีจุดเด่นและข้อดีเฉพาะที่เหมาะสมกับบริบทที่แตกต่างกัน

1.1.4 ภาษาการเขียนโปรแกรม (s)

ภาษาการเขียนโปรแกรมคือภาษาทางการที่ประกอบด้วยชุดคำสั่งที่สามารถใช้ในการสร้างผลลัพธ์ต่าง ๆ ได้ โปรแกรมเมอร์ใช้ภาษาเหล่านี้ในการเขียนโปรแกรมเพื่อให้คอมพิวเตอร์ดำเนินการตามภารกิจที่กำหนด แต่ละภาษาจะมีไวยากรณ์ (syntax) และความหมาย (semantics) ของคำสั่งเฉพาะตัว บทนี้จะช่วยให้ผู้เริ่มต้นเข้าใจพื้นฐานของภาษาการเขียนโปรแกรม

ภาษาการเขียนโปรแกรมคืออะไร? ภาษาการเขียนโปรแกรมคือเครื่องมือที่ช่วยให้นักพัฒนาซอฟต์แวร์สามารถสื่อสารกับคอมพิวเตอร์และสร้างแอปพลิเคชันต่าง ๆ ได้ ภาษาเหล่านี้ช่วยให้โปรแกรมเมอร์สามารถเขียนคำสั่งในรูปแบบที่มนุษย์อ่านได้ แล้วแปลงคำสั่งเหล่านั้นให้กลายเป็นรหัสเครื่องที่คอมพิวเตอร์สามารถประมวลผลได้

ประเภทของภาษาการเขียนโปรแกรม ภาษาการเขียนโปรแกรมแบ่งออกเป็นสองประเภทหลัก คือ ภาษาในระดับสูง (High-Level) และภาษาในระดับต่ำ (Low-Level)

1. **ภาษาในระดับสูง (High-Level Languages):**
    - ภาษาเหล่านี้มีลักษณะใกล้เคียงกับภาษามนุษย์ และอ่านเขียนได้ง่าย
    - ไม่จำเป็นต้องมีความรู้เฉพาะทางเกี่ยวกับสถาปัตยกรรมคอมพิวเตอร์
    - ตัวอย่างเช่น Python, Java, C++, JavaScript, และ Ruby

    ตัวอย่างของภาษาในระดับสูง (Python):

```python
#HelloWorld.py
print("Hello, World!")
```

Listing 1.5: Example of High-Level Language (Python)

2. **ภาษาในระดับต่ำ (Low-Level Languages):**
    - ภาษาเหล่านี้มีลักษณะใกล้เคียงกับภาษาเครื่อง (binary) และอ่านเขียนได้ยาก
    - มีความเกี่ยวข้องโดยตรงกับสถาปัตยกรรมของเครื่องคอมพิวเตอร์
    - ตัวอย่างเช่น ภาษาแอสเซมบลี (Assembly) และรหัสเครื่อง (Machine Code)

    ตัวอย่างของภาษาในระดับต่ำ (Assembly):

```assembly
MOV AX, 4C00h
INT 21h
```

Listing 1.6: Example of High-Level Language (Assembly)

ตัวอย่างภาษาการเขียนโปรแกรมยอดนิยม:

1. **Python:**
    - เป็นที่รู้จักในเรื่องความอ่านง่าย และความเรียบง่าย
    - ใช้งานกว้างขวางในด้านพัฒนาเว็บไซต์ วิเคราะห์ข้อมูล ปัญญาประดิษฐ์ และระบบอัตโนมัติ
    - ตัวอย่างไวยากรณ์:

```python
#GreetFunc.py
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))
```

Listing 1.7: Example code of Python

2. **Java:**
    - เป็นภาษาเชิงวัตถุ (Object-Oriented) และออกแบบให้ใช้งานได้ข้ามแพลตฟอร์ม
    - นิยมใช้ในการพัฒนาแอปพลิเคชันองค์กร และแอปมือถือบนระบบ Android
    - ตัวอย่างไวยากรณ์:

```java
//HelloWorld.java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Listing 1.8: Example code of Java

3. **C++:**

   - เป็นส่วนขยายของภาษา C โดยเพิ่มคุณสมบัติของการเขียนเชิงวัตถุ
   - ใช้ในการพัฒนาระบบ เกม และการจำลองแบบเรียลไทม์
   - ตัวอย่างไวยากรณ์:

```cpp
//HelloWorld.cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

Listing 1.9: Example code of C++

4. **JavaScript:**

   - ใช้หลัก ๆ ในการพัฒนาเว็บเพจแบบอินเทอร์แอคทีฟ และส่วนติดต่อผู้ใช้ (Frontend)
   - สามารถใช้งานฝั่งเซิร์ฟเวอร์ได้ด้วย เช่น Node.js
   - ตัวอย่างไวยากรณ์:

```javascript
//GreetFunc.js
function greet(name) {
    return `Hello, ${name}!`;
}

console.log(greet("Alice"));
```

Listing 1.10: Example code of Javascript

ทำไมควรเรียนรู้ภาษาการเขียนโปรแกรมหลายภาษา?

- **ความหลากหลายในการใช้งาน:** ภาษาต่าง ๆ เหมาะกับงานที่แตกต่างกัน เช่น Python เหมาะกับการวิเคราะห์ข้อมูล ขณะที่ C++ เหมาะกับงานที่ต้องการประสิทธิภาพสูง
- **โอกาสในอาชีพ:** การรู้หลายภาษาช่วยให้มีความยืดหยุ่น และสามารถเข้าทำงานได้ในหลายสายงาน
- **ทักษะการแก้ปัญหา:** แต่ละภาษาสอนมุมมองการคิดที่แตกต่างกัน ช่วยเสริมสร้างความสามารถในการแก้ปัญหา

แนวทางการเลือกภาษาการเขียนโปรแกรม:

- **ผู้เริ่มต้น:** แนะนำให้เริ่มต้นด้วย Python เนื่องจากอ่านง่ายและเข้าใจได้เร็ว
- **การพัฒนาเว็บ:** ใช้ HTML, CSS, JavaScript และเฟรมเวิร์ก เช่น React หรือ Angular
- **การพัฒนาแอปมือถือ:** ใช้ Swift สำหรับ iOS และ Kotlin สำหรับ Android
- **การพัฒนาระบบ:** ใช้ C หรือ C++ สำหรับแอปที่ต้องการประสิทธิภาพสูง

โดยสรุป ภาษาการเขียนโปรแกรมเป็นเครื่องมือสำคัญในการสร้างซอฟต์แวร์ โดยแต่ละภาษามีจุดเด่นและการใช้งานที่เฉพาะตัว การเข้าใจภาษาต่าง ๆ และวิธีการใช้งาน จะช่วยให้โปรแกรมเมอร์สามารถเลือกใช้เครื่องมือที่เหมาะสมกับภารกิจ และพัฒนาทักษะการแก้ปัญหาได้อย่างมีประสิทธิภาพ

1.2 ภาษาไพธอนคืออะไร (What is Python?)

ไพธอน (Python) [2] เป็นภาษาการเขียนโปรแกรมระดับสูงที่ทำงานแบบแปลผล (interpreted) ซึ่งขึ้นชื่อเรื่องความเรียบง่ายและอ่านเข้าใจได้ง่าย ไพธอนรองรับหลายแนวทางการเขียนโปรแกรม เช่น กระบวนวิธี (procedural), เชิงวัตถุ (object-oriented) และเชิงฟังก์ชัน (functional programming)

ด้วยโครงสร้างที่เรียบง่ายและไวยากรณ์ที่ชัดเจน ทำให้ไพธอนเป็นภาษาที่เหมาะอย่างยิ่งสำหรับผู้เริ่มต้นเรียนรู้การเขียนโปรแกรม นอกจากนี้ ไพธอนยังมีไลบรารีและเฟรมเวิร์กจำนวนมากที่นิยมใช้ในการพัฒนาเว็บไซต์ การวิเคราะห์ข้อมูล ปัญญาประดิษฐ์ และการคำนวณทางวิทยาศาสตร์

คุณลักษณะเด่นของไพธอน:

- **ความสามารถในการอ่านง่าย (Readability):** โครงสร้างคำสั่งที่อ่านง่าย
- **ไลบรารีที่ครอบคลุม (Extensive Libraries):** มีไลบรารีและเฟรมเวิร์กมากมาย
- **ความยืดหยุ่น (Versatility):** ใช้ได้ในหลายด้าน เช่น เว็บ, ข้อมูล, AI, งานอัตโนมัติ
- **การสนับสนุนจากชุมชน (Community Support):** มีชุมชนผู้ใช้ขนาดใหญ่และช่วยเหลือกันดี

ประวัติของภาษาไพธอน (History of Python) Guido van Rossum เป็นผู้สร้างภาษาไพธอน และเปิดตัวครั้งแรกในปี ค.ศ. 1991 โดยออกแบบมาเพื่อเน้นความเรียบง่ายและความสามารถในการอ่านของโค้ด ปัจจุบันไพธอนได้เติบโตขึ้นจนกลายเป็นหนึ่งในภาษาการเขียนโปรแกรมที่ได้รับความนิยมมากที่สุดในโลก

การใช้งานของไพธอน (Python Applications)

- **การพัฒนาเว็บ:** Django, Flask
- **วิทยาการข้อมูล:** Pandas, NumPy, Matplotlib
- **การเรียนรู้ของเครื่อง:** TensorFlow, PyTorch
- **งานอัตโนมัติ:** สคริปต์สำหรับการทำงานอัตโนมัติ

1.3 การติดตั้งภาษาไพธอน (Installing Python)

คู่มือการติดตั้งแบบทีละขั้นตอน:

ดาวน์โหลดไพธอน (Download Python):

1. เข้าเว็บไซต์ทางการของไพธอน: python.org
2. ไปที่หน้าดาวน์โหลด และเลือกเวอร์ชันที่เหมาะสมกับระบบปฏิบัติการของคุณ (Windows, macOS, Linux)

เรียกใช้งานตัวติดตั้ง (Run the Installer):

- สำหรับ Windows: ดาวน์โหลดไฟล์ติดตั้งแบบ .exe แล้วรัน อย่าลืมติ๊กช่อง “Add Python to PATH” ก่อนดำเนินการต่อ
- สำหรับ macOS: ดาวน์โหลดไฟล์ .pkg แล้วติดตั้ง
- สำหรับ Linux: ใช้โปรแกรมจัดการแพ็กเกจติดตั้ง เช่น **sudo apt-get install python3** สำหรับ Ubuntu

ตรวจสอบการติดตั้ง (Verify Installation):

1. เปิด terminal หรือ command prompt
2. พิมพ์คำสั่ง **python –version** หรือ **python3 –version** เพื่อตรวจสอบเวอร์ชันที่ติดตั้ง

การตั้งค่าสภาพแวดล้อมสำหรับพัฒนาโปรแกรม (Setting Up a Development Environment):

- **โปรแกรมพัฒนาแบบครบวงจร (IDE):** เป็นโปรแกรมที่ช่วยในการพัฒนา เช่น PyCharm, Visual Studio Code, หรือ Jupyter Notebook
- **ตัวแก้ไขโค้ด (Code Editors):** เครื่องมือที่เบาและเหมาะสำหรับการเขียนโค้ด เช่น Sublime Text, Atom

แนะนำเครื่องมือ IDE และ Editor ที่ควรใช้:

- **Visual Studio Code:** ตัวแก้ไขโค้ดที่เบาแต่ทรงพลัง พร้อมปลั๊กอินรองรับ Python
- **PyCharm:** IDE สำหรับ Python โดยเฉพาะ มีฟีเจอร์ครบถ้วน เช่น วิเคราะห์โค้ด, ดีบัก, การควบคุมเวอร์ชัน
- **Jupyter Notebook:** แอปพลิเคชันเว็บที่เปิดให้สร้างและแบ่งปันโค้ด แผนภาพ สมการ และข้อความอธิบาย

การเขียนโปรแกรมไพธอนแรกของคุณ (Writing Your First Python Program):

1. เปิด IDE หรือ editor ที่คุณเลือก แล้วพิมพ์โค้ดต่อไปนี้:

```
#HelloWorld.py
print("Hello, World!")
```

Listing 1.11: First Python program HelloWorld.py

2. บันทึกไฟล์ด้วยนามสกุล **.py** (เช่น **HelloWorld.py**)

วิธีรันโปรแกรม:

1. เปิด terminal หรือ command prompt
2. ไปยังโฟลเดอร์ที่คุณบันทึกไฟล์ไว้
3. พิมพ์คำสั่ง **python hello.py** แล้วกด Enter

```
Hello, World!
```

Listing 1.12: Output of program HelloWorld.py

ขอแสดงความยินดี! คุณได้เขียนและรันโปรแกรมไพธอนโปรแกรมแรกของคุณแล้ว

เนื้อหาในบทนี้ได้นำเสนอข้อมูลเบื้องต้นเกี่ยวกับการเขียนโปรแกรมและภาษาไพธอนอย่างครบถ้วน [3] ครอบคลุมทั้งแนวคิดพื้นฐาน ประวัติศาสตร์ การใช้งาน ตลอดจนขั้นตอนเชิงปฏิบัติในการติดตั้ง Python และเขียนโปรแกรมแรก

การเขียนโปรแกรมคือการออกแบบและสร้างโปรแกรมคอมพิวเตอร์ที่สามารถทำงานได้จริง เพื่อตอบสนองต่อภารกิจเฉพาะทางด้านคอมพิวเตอร์ โดยการเขียนโค้ดในภาษาที่คอมพิวเตอร์สามารถเข้าใจและประมวลผลได้ กระบวนการนี้เป็นสิ่งสำคัญในยุคดิจิทัล เพราะช่วยควบคุมการทำงานระหว่างมนุษย์และเครื่องจักร ทำงานอัตโนมัติ และแก้ปัญหาซับซ้อนอย่างมีประสิทธิภาพ หลักการเขียนโปรแกรมได้นำไปประยุกต์ใช้กับเทคโนโลยีใหม่ ๆ เช่น ปัญญาประดิษฐ์ การรู้จำเสียง และอินเทอร์เน็ตของสรรพสิ่ง (IoT) ทำให้เป็นทักษะสำคัญสำหรับทั้งมืออาชีพและบุคคลทั่วไป ผู้ที่มีความชำนาญด้านภาษาโปรแกรมสามารถนำไอเดียไปพัฒนาเป็นแอปพลิเคชัน และสร้างนวัตกรรมใหม่ ๆ ที่ตอบโจทย์ความท้าทายในโลกแห่งความจริง นอกจากนี้ การเขียนโปรแกรมยังช่วยเสริมสร้างการคิดเชิงคำนวณ ทักษะการแก้ปัญหา และความคิดสร้างสรรค์ ซึ่งล้วนเป็นพื้นฐานสำคัญของนวัตกรรมและการเป็นผู้ประกอบการ

บทที่ 1 โจทย์และแบบฝึกหัด: บทนำสู่การเขียนโปรแกรมและภาษาไพธอน

1.1 เข้าใจอัลกอริทึม
ให้นิยามอัลกอริทึม และยกตัวอย่างอัลกอริทึมในการชงชา 1 ถ้วย

1.2 เขียนซอร์สโค้ดอย่างง่าย
เขียนโปรแกรมภาษา Python อย่างง่ายเพื่อคำนวณผลรวมของตัวเลขสองจำนวนที่ผู้ใช้ป้อน

1.3 คอมไพล์กับการแปลผล
อธิบายความแตกต่างระหว่างการคอมไพล์และการแปลผล พร้อมยกตัวอย่างประกอบ

1.4 จำแนกภาษาการเขียนโปรแกรม
จำแนกภาษาต่อไปนี้ว่าเป็นภาษาระดับสูงหรือระดับต่ำ: Python, Assembly, C++, และ JavaScript

1.5 การติดตั้งโปรแกรม Python
อธิบายขั้นตอนการติดตั้ง Python บนระบบปฏิบัติการ Windows

1.6 โปรแกรม Python แรก
เขียนโปรแกรม Python เพื่อแสดงข้อความว่า "Welcome to Python Programming!"

1.7 คุณลักษณะของ Python
ระบุคุณลักษณะเด่นของ Python จำนวน 4 ข้อ พร้อมคำอธิบายโดยสังเขป

1.8 เขียนอัลกอริทึม
สร้างอัลกอริทึมเพื่อหาค่ามากที่สุดของจำนวน 3 จำนวน

1.9 การใช้งาน Python
ระบุ 3 ด้านที่มีการใช้ Python อย่างแพร่หลาย พร้อมยกตัวอย่างไลบรารีหรือเฟรมเวิร์กสำหรับแต่ละด้าน

1.10 การตั้งค่าสภาพแวดล้อมพัฒนา
อธิบายความแตกต่างระหว่างการคอมไพล์และการแปลผล พร้อมยกตัวอย่าง

ภาคผนวก

A1.1 แหล่งเรียนรู้เพิ่มเติม

หนังสือ:

- *Python Crash Course* โดย Eric Matthes
- *Automate the Boring Stuff with Python* โดย Al Sweigart
- *Learning Python* โดย Mark Lutz

บทเรียนออนไลน์:

- Python.org
- W3Schools Python Tutorial
- Real Python

คอร์สเรียน:

- Coursera: Python for Everybody – มหาวิทยาลัยมิชิแกน
- edX: แนะนำวิทยาการคอมพิวเตอร์ด้วย Python – MIT
- Udacity: แนะนำการเขียนโปรแกรม Python

A1.2 เอกสารอ้างอิง

- Python Software Foundation. (2024). เอกสาร Python. https://docs.python.org/3/
- Van Rossum, G., & Drake Jr, F. L. (2009). *คู่มืออ้างอิง Python 3*. CreateSpace.
- Downey, A. B. (2015). *Think Python: คิดแบบนักวิทยาการคอมพิวเตอร์*. O'Reilly Media.

A1.3 แบบฝึกหัดเพิ่มเติม

แบบฝึกหัดที่ 1: เข้าใจอัลกอริทึม

- เขียนอัลกอริทึมเพื่อหาค่ามากที่สุดในลิสต์ของตัวเลข
- เขียนอัลกอริทึมเรียงลำดับลิสต์จำนวนเต็มจากน้อยไปมาก

แบบฝึกหัดที่ 2: ฝึกเขียน Python

- เขียนโปรแกรม Python เพื่อหาค่ากำลังสองของจำนวนที่ผู้ใช้ป้อน
- เขียนสคริปต์ Python เพื่อตรวจสอบว่าตัวเลขที่ป้อนเป็นเลขคู่หรือเลขคี่

แบบฝึกหัดที่ 3: วิเคราะห์โค้ด

- อ่านและอธิบายว่าโค้ด Python ด้านล่างทำงานอย่างไร:

โค้ด Python:

```python
def fibonacci(n):
    a, b = 0, 1
    for i in range(2, n):
        a, b = b, a + b
    return b

print(fibonacci(10))
```

Listing 1.13: โค้ด Python สำหรับลำดับฟีโบนักชีตัวที่ 10

A1.4 ตัวอย่างเชิงปฏิบัติ

ตัวอย่างที่ 1: โปรแกรม Python อย่างง่าย

```python
#
x = input("Enter the first number: ")
y = input("Enter the second number: ")

sum = int(x) + int(y)
print("The sum is:", sum)
```

Listing 1.14: โปรแกรม Python เพื่อบวกเลขสองจำนวนที่ป้อน

ตัวอย่างที่ 2: การใช้ฟังก์ชัน

```python
#

def max_of_two(a, b):
    return a if a > b else b

n1 = int(input("Enter first number: "))
n2 = int(input("Enter second number: "))
print("Greater number is:", max_of_two(n1, n2))
```

Listing 1.15: หาค่ามากสุดระหว่างสองตัวเลขด้วยฟังก์ชัน

บทที่ 2

ตัวแปร ชนิดข้อมูล และการรับข้อมูลจากผู้ใช้

บทนี้นำเสนอภาพรวมที่ครอบคลุมเกี่ยวกับตัวแปร ชนิดข้อมูล และการรับข้อมูลจากผู้ใช้ในภาษา Python เพื่อวางรากฐานความเข้าใจที่จำเป็นสำหรับการเขียนโปรแกรม ผู้เรียนจะได้เข้าใจว่า Python จัดการและประมวลผลข้อมูลอย่างไร โดยการศึกษาการประกาศตัวแปร การตั้งชื่อตัวแปรตามหลักเกณฑ์ และการจัดการชนิดข้อมูลต่าง ๆ เช่น จำนวนเต็ม (integers), ทศนิยม (floats), และจำนวนเชิงซ้อน (complex numbers) ซึ่งแสดงให้เห็นถึงความสามารถของ Python ในการจัดการข้อมูลอย่างหลากหลาย นอกจากนี้ การสำรวจชนิดข้อมูลแบบลำดับ เช่น สตริง (strings), ลิสต์ (lists), และทูเพิล (tuples) ยังแสดงถึงความยืดหยุ่นในการจัดการข้อมูลของ Python อีกด้วย

บทนี้ยังกล่าวถึงชนิดข้อมูลบูลีนและดิกชันนารี ซึ่งช่วยเสริมความเข้าใจในการดำเนินการทางตรรกะและการใช้โครงสร้างข้อมูลแบบคู่คีย์-ค่า โดยมีตัวอย่างการใช้งานจริงและแบบฝึกหัดตลอดบท เพื่อส่งเสริมการเรียนรู้แบบลงมือทำและช่วยให้เข้าใจแนวคิดเชิงทฤษฎีได้ดีขึ้น ตัวอย่างการประยุกต์ เช่น การคำนวณพื้นที่ของสี่เหลี่ยมผืนผ้าโดยใช้ตัวแปร ชนิดข้อมูล และการรับข้อมูลจากผู้ใช้ ถือเป็นการนำแนวคิดต่าง ๆ มาผสมผสานเพื่อแก้ปัญหาในโลกจริงอย่างเป็นรูปธรรม

2.1 วงจรการพัฒนาโปรแกรม

วงจรการพัฒนาโปรแกรมคือกระบวนการที่มีโครงสร้างชัดเจน เพื่อช่วยให้นักพัฒนาเขียนโปรแกรมอย่างเป็นระบบ ซึ่งจะช่วยให้โปรแกรมที่พัฒนาขึ้นมีความน่าเชื่อถือและมีประสิทธิภาพมากยิ่งขึ้น ขั้นตอนหลักของวงจรการพัฒนาโปรแกรมมีดังนี้:

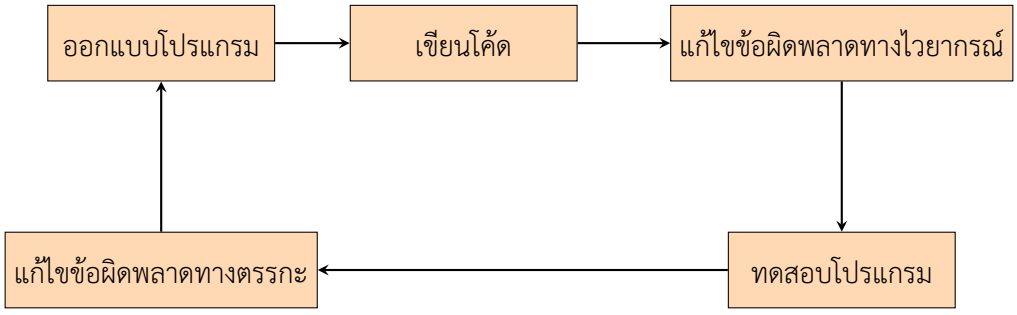

Figure 2.1: วงจรการพัฒนาโปรแกรม

วงจรการพัฒนาโปรแกรม

1. **ออกแบบโปรแกรม (Design the Program)**

   ขั้นแรกคือการออกแบบโปรแกรม โดยใช้เครื่องมือ เช่น:

- **รหัสเทียม (Pseudocode):** เขียนลำดับขั้นตอนของโปรแกรมเป็นภาษาธรรมดา เพื่อช่วยวางโครงสร้างก่อนลงมือเขียนโค้ดจริง
- **ผังงาน (Flowchart):** แสดงลำดับการทำงานของโปรแกรมเป็นภาพ ทำให้เห็นภาพรวมของตรรกะและขั้นตอนต่าง ๆ
- **ตาราง IPO (Input-Process-Output):** ใช้กำหนดข้อมูลนำเข้า (Input) การประมวลผล (Process) และผลลัพธ์ (Output) อย่างชัดเจน

2. **เขียนโค้ด (Write the Code)**

   เมื่อลงรายละเอียดในการออกแบบเรียบร้อยแล้ว ขั้นตอนถัดไปคือการเขียนโค้ด โดยอ้างอิงจาก pseudocode และ flowchart ที่ได้เตรียมไว้

3. **แก้ไขข้อผิดพลาดทางไวยากรณ์ (Correct Syntax Errors)**

   เมื่อเขียนโค้ดเสร็จแล้วต้องตรวจสอบข้อผิดพลาดทางไวยากรณ์ (Syntax Errors) เช่น วงเล็บไม่ครบ พิมพ์ผิด ใช้คำสงวนผิดที่ เป็นต้น

4. **ทดสอบโปรแกรม (Test the Program)**

   ทดสอบโปรแกรมเพื่อให้แน่ใจว่าโปรแกรมทำงานได้ตามที่ออกแบบไว้ การทดสอบมีหลายระดับ เช่น:

   - **Unit Testing:** ทดสอบฟังก์ชันหรือหน่วยย่อยของโปรแกรมแยกจากกัน
   - **Integration Testing:** ทดสอบการทำงานร่วมกันของหลาย ๆ ส่วน
   - **System Testing:** ทดสอบโปรแกรมทั้งระบบให้ทำงานตามข้อกำหนดจริง

5. **แก้ไขข้อผิดพลาดทางตรรกะ (Correct Logic Errors)**

   ข้อผิดพลาดที่ไม่ได้ขัดต่อไวยากรณ์ แต่ทำให้ผลลัพธ์ไม่ถูกต้อง เช่น การใช้เครื่องหมายผิด หรือเงื่อนไขผิด จะต้องใช้เทคนิคการดีบักและตรวจสอบตรรกะให้ถูกต้อง

ความสำคัญของวงจรการพัฒนาโปรแกรม

- **เพิ่มคุณภาพโปรแกรม:** ช่วยตรวจจับปัญหาแต่เนิ่น ๆ ลดข้อผิดพลาด
- **ดูแลและพัฒนาต่อได้ง่าย:** โครงสร้างที่ดีช่วยให้ปรับปรุงและอัปเดตโค้ดในอนาคตได้ง่าย
- **เพิ่มความน่าเชื่อถือ:** การทดสอบและแก้ไขอย่างละเอียดทำให้โปรแกรมทำงานได้เสถียร
- **เสริมการทำงานเป็นทีม:** เครื่องมืออย่าง pseudocode และ flowchart ช่วยให้สมาชิกในทีมเข้าใจร่วมกันได้ง่าย

วงจรการพัฒนาโปรแกรมจึงเป็นโครงสร้างพื้นฐานที่สำคัญในการสร้างซอฟต์แวร์คุณภาพสูง ที่สามารถใช้งานได้จริง มีประสิทธิภาพ และดูแลต่อยอดได้ในระยะยาว

2.1.1 รหัสเทียม/อัลกอริทึม (Pseudocode/Algorithm)

ก่อนจะเริ่มเขียนโค้ดจริง การร่างลำดับตรรกะของโปรแกรมด้วยรหัสเทียมหรืออัลกอริทึม (pseudocode) เป็นสิ่งที่มีประโยชน์อย่างยิ่ง รหัสเทียมเป็นการเขียนขั้นตอนการทำงานของโปรแกรมในรูปแบบภาษาธรรมดา ช่วยในการวางแผนโครงสร้างและลำดับขั้นตอนของโปรแกรมโดยไม่ต้องกังวลเรื่องไวยากรณ์ ทำให้มั่นใจได้ว่าตรรกะของโปรแกรมถูกต้องและชัดเจน ตัวอย่างด้านล่างเป็นรหัสเทียมของโปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้า:

Algorithm 1: Algorithm to Calculate the Area of a Rectangle

Input: Width and height of the rectangle
Output: The area of the rectangle

```
1 begin
2     Print "Enter the width of the rectangle:";
3     Read width;
4     Print "Enter the height of the rectangle:";
5     Read height;
6     area = width * height;
7     Print "The area of the rectangle is", area;
```

ประโยชน์ของการใช้รหัสเทียม (Pseudocode)

- **ชัดเจน:** รหัสเทียมแสดงตรรกะของโปรแกรมอย่างเข้าใจง่าย เป็นมิตรกับมนุษย์
- **ไม่ขึ้นกับภาษาใด:** ไม่ผูกกับไวยากรณ์ของภาษาใดภาษาเดียว ทำให้เน้นที่ตรรกะเป็นหลัก
- **ช่วยวางแผน:** ใช้ในการร่างโครงสร้างก่อนลงมือเขียนโค้ดจริง ช่วยลดข้อผิดพลาดและทำให้โค้ดเป็นระเบียบ
- **ช่วยในการสื่อสาร:** ใช้ร่วมกันในทีมเพื่ออธิบายแนวคิดกับคนที่ไม่เชี่ยวชาญด้านเขียนโปรแกรม

การใช้รหัสเทียมจึงเป็นขั้นตอนสำคัญที่ช่วยให้โปรแกรมมีโครงสร้างที่ดี เข้าใจง่าย และลดปัญหาในการเขียนโค้ดจริง ช่วยให้ซอฟต์แวร์มีความน่าเชื่อถือและดูแลรักษาได้ในระยะยาว

2.1.2 ผังงาน (Flow Chart)

การสร้างผังงานเป็นวิธีที่มีประสิทธิภาพในการแสดงตรรกะของโปรแกรมในเชิงภาพ ผังงานช่วยให้เห็นภาพรวมของกระบวนการทำงาน การรับข้อมูล การประมวลผล การแสดงผล และจุดตัดสินใจของโปรแกรม โดยใช้สัญลักษณ์มาตรฐาน ทำให้เข้าใจโครงสร้างของโปรแกรมได้ง่ายขึ้นและสามารถสื่อสารแนวคิดกับผู้อื่นได้อย่างชัดเจน ตัวอย่างต่อไปนี้คือผังงานของโปรแกรมที่ใช้คำนวณพื้นที่ของสี่เหลี่ยมผืนผ้า:

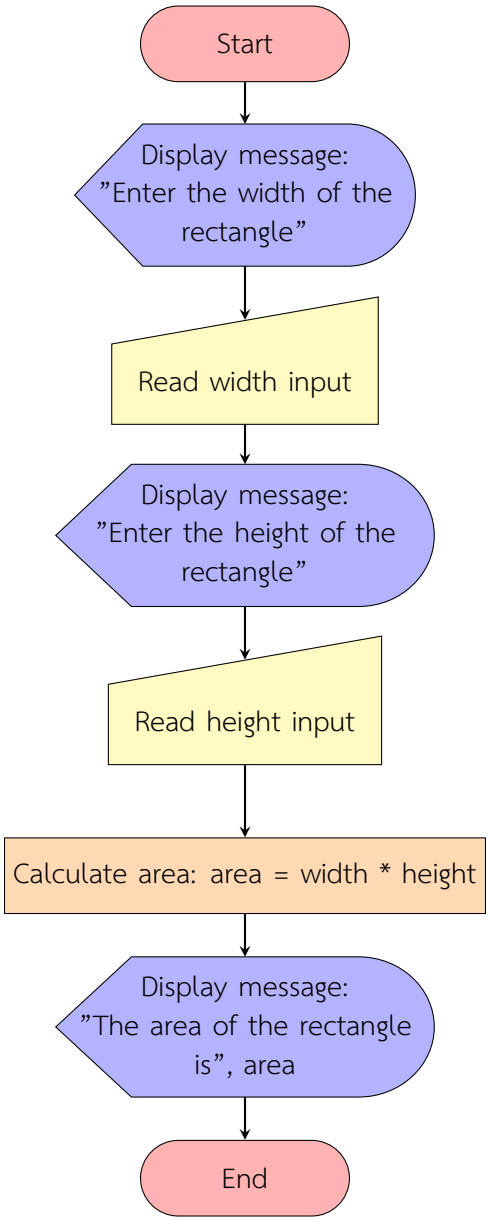

Figure 2.2: ผังงานของโปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้า

สัญลักษณ์ในผังงาน (Flowchart Symbols)

- **วงรี (Start/End):** แทนจุดเริ่มต้นและจุดสิ้นสุดของโปรแกรม
- **สี่เหลี่ยมขนมเปียกปูน (Input/Output):** แทนการรับค่าหรือแสดงผล เช่น รับข้อมูลจากผู้ใช้หรือแสดงข้อความ
- **สี่เหลี่ยมผืนผ้า (Process):** แทนขั้นตอนการประมวลผล เช่น การคำนวณหรือการดำเนินการต่าง ๆ

ประโยชน์ของการใช้ผังงาน (Benefits of Using Flowcharts)

- **มองภาพรวมได้ง่าย:** ผังงานแสดงขั้นตอนของโปรแกรมในเชิงภาพ ทำให้เข้าใจลำดับการทำงานได้ชัดเจน
- **ใช้ในการสื่อสาร:** เป็นเครื่องมือสื่อสารที่ดีระหว่างทีมพัฒนาและผู้เกี่ยวข้อง
- **ช่วยหาจุดผิดพลาด:** ผังงานช่วยให้ระบุข้อผิดพลาดทางตรรกะได้ง่าย
- **ใช้เป็นเอกสารอ้างอิง:** บันทึกการออกแบบโปรแกรมเพื่อใช้อ้างอิงหรือปรับปรุงในอนาคต

ผังงานช่วยเสริมรหัสเทียมโดยให้มุมมองเชิงภาพของโครงสร้างและการทำงานของโปรแกรม ช่วยให้การวางแผนและการสื่อสารในวงจรการพัฒนาโปรแกรมมีประสิทธิภาพยิ่งขึ้น

2.1.3 แผนภูมิ IPO (Input Process Output)

แผนภูมิ IPO เป็นเครื่องมือที่มีประโยชน์ในขั้นตอนออกแบบโปรแกรม เพื่อกำหนดโครงสร้างของโปรแกรมอย่างชัดเจน โดยระบุว่าโปรแกรมต้องรับข้อมูลใด (Input) ประมวลผลอย่างไร (Process) และให้ผลลัพธ์ใด (Output) ซึ่งช่วยให้เข้าใจหน้าที่ของโปรแกรมได้อย่างครบถ้วน และจัดโครงสร้างโค้ดได้อย่างเป็นระบบ ตัวอย่างต่อไปนี้เป็นแผนภูมิ IPO สำหรับโปรแกรมคำนวณพื้นที่ของสี่เหลี่ยมผืนผ้า:

Table 2.1: ตัวอย่างแผนภูมิ IPO

Input	Process	Output
Width of the rectangle	Multiply width by height	Area of the rectangle
Height of the rectangle		

จากตารางจะเห็นว่า Input คือความกว้างและความสูงของสี่เหลี่ยมผืนผ้า ส่วน Process คือการคูณค่าทั้งสอง และ Output คือพื้นที่ของสี่เหลี่ยม การใช้แผนภูมิ IPO ควบคู่กับรหัสเทียมและผังงานช่วยให้การออกแบบโปรแกรมเป็นระบบ ครอบคลุมทุกขั้นตอน และจัดทำเอกสารได้อย่างครบถ้วน

2.1.4 เขียนโปรแกรมจากอัลกอริทึม (Program From Algorithm)

ขั้นตอนการแปลงอัลกอริทึมหรือผังงานก่อนหน้าให้เป็นโปรแกรมภาษา Python มีดังนี้: เริ่มจากรับค่าความกว้างและความสูงของสี่เหลี่ยมผืนผ้าจากผู้ใช้โดยใช้คำสั่ง `input()` แล้วแปลงเป็นเลขทศนิยมด้วย `float()` จากนั้นคำนวณพื้นที่โดยคูณค่าทั้งสอง และแสดงผลลัพธ์ด้วย `print()` ซึ่งการดำเนินการตามลำดับนี้จะช่วยให้โค้ดมีความถูกต้องและสอดคล้องกับตรรกะของอัลกอริทึม

```python
# Python program to calculate the area of a rectangle

# Prompt the user to enter the width of the rectangle
width = float(input("Enter the width of the rectangle: "))

# Prompt the user to enter the height of the rectangle
height = float(input("Enter the height of the rectangle: "))

# Calculate the area
area = width * height

# Print the area of the rectangle
print("The area of the rectangle is:", area)
```

Listing 2.1: Python program to calculate the area of a rectangle

โปรแกรมนี้ทำงานตามขั้นตอนของอัลกอริทึม:

- รับค่าความกว้างและความสูงจากผู้ใช้
- คำนวณพื้นที่โดยใช้สูตร `width * height`
- แสดงผลลัพธ์ที่คำนวณได้ออกทางหน้าจอ

2.2 ตัวแปร ()

ตัวแปรในภาษาโปรแกรมเปรียบเสมือนกล่องที่มีป้ายชื่อซึ่งใช้เก็บข้อมูล ชื่อของตัวแปรทำหน้าที่เป็นฉลากที่ช่วยให้สามารถเข้าถึงและแก้ไขข้อมูลภายในได้อย่างสะดวก ตัวแปรมีความยืดหยุ่น ไม่เหมือนค่าคงที่ซึ่งเปลี่ยนไม่ได้ ตัวแปรสามารถเปลี่ยนแปลงค่าระหว่างการทำงานของโปรแกรมได้ ความยืดหยุ่นนี้ทำให้โปรแกรมสามารถปรับตัวตามสถานการณ์ต่าง ๆ และสามารถประมวลผลข้อมูลได้หลากหลาย จึงช่วยให้สร้างโปรแกรมที่แข็งแรงและปรับใช้ได้ง่าย กล่าวโดยสรุป ตัวแปรช่วยให้นักพัฒนาสามารถจัดการข้อมูลได้อย่างมีประสิทธิภาพ ซึ่งเป็นรากฐานสำคัญของแนวทางการเขียนโปรแกรมสมัยใหม่

แนวคิดสำคัญ (Key Concepts):

- **การประกาศตัวแปร (Variable Declaration):** การกำหนดค่าหรือตั้งค่าตัวแปร
- **การตั้งชื่อตัวแปร (Variable Naming):** การเลือกชื่อตัวแปรที่เหมาะสมตามกฎและแนวทางที่กำหนดไว้

2.2.1 การประกาศและกำหนดค่าตัวแปรในภาษา Python

ในภาษา Python เราสามารถประกาศและกำหนดค่าตัวแปรได้โดยใช้เครื่องหมายเท่ากับ ('=')

ตัวอย่างการประกาศและกำหนดค่าตัวแปร:

```
# Declaring variables
age = 25
name = "Alice"
height = 5.6
```

Listing 2.2: Example of Declaring and Assigning Variables in Python

ตัวอย่างนี้แสดงการกำหนดค่าตัวแปรทั้งสามตัว ได้แก่ **age**, **name**, และ **height** ซึ่งเก็บข้อมูลประเภทจำนวนเต็ม (integer), สตริง (string) และจำนวนทศนิยม (float) ตามลำดับ ตัวแปรเหล่านี้สามารถนำไปใช้ในโปรแกรมเพื่อจัดเก็บ เรียกใช้ และประมวลผลข้อมูลต่อไปได้

2.2.2 กฎการตั้งชื่อตัวแปร (Variable Naming Rules)

การตั้งชื่อตัวแปรใน Python มีความสำคัญต่อความอ่านง่าย และการดูแลรักษาโค้ดให้มีประสิทธิภาพ ตัวแปรต้องเริ่มต้นด้วยตัวอักษรหรือตัวขีดล่าง (_) และสามารถมีตัวอักษร ตัวเลข และขีดล่างร่วมด้วยได้ ชื่อตัวแปรมีลักษณะแยกแยะตัวพิมพ์เล็กและใหญ่ (case-sensitive) เช่น **age** กับ **Age** ถือว่าเป็นคนละตัวกัน ควรตั้งชื่อให้สื่อความหมาย ชัดเจน และหลีกเลี่ยงการใช้คีย์เวิร์ดของ Python เป็นชื่อตัวแปร เพื่อป้องกันความสับสนหรือข้อผิดพลาดจากไวยากรณ์ นอกจากนี้ ควรใช้รูปแบบที่สม่ำเสมอ เช่น การใช้ตัวพิมพ์เล็กและขีดล่าง (*snake_case*) ในชื่อตัวแปรที่ประกอบด้วยหลายคำ

- ตัวแปรต้องขึ้นต้นด้วยตัวอักษรหรือตัวขีดล่าง (’_’)
- สามารถมีตัวอักษร ตัวเลข และขีดล่างได้
- ชื่อตัวแปรแยกแยะตัวพิมพ์เล็ก/ใหญ่ (**age** และ **Age** เป็นคนละตัว)
- ควรตั้งชื่อให้สื่อถึงวัตถุประสงค์ของข้อมูล

ณ Python รุ่น 3.10 มีคีย์เวิร์ดที่สงวนไว้จำนวน 35 คำ ได้แก่:

Table 2.2: คำสงวน (Keywords) ในภาษา Python

False	None	True	and	as
assert	async	await	break	class
continue	def	del	elif	else
except	finally	for	from	global
if	import	in	is	lambda
nonlocal	not	or	pass	raise
return	try	while	with	yield

คีย์เวิร์ดเหล่านี้เป็นคำสำคัญของภาษา Python ซึ่งมีความหมายเฉพาะ และไม่สามารถนำมาใช้เป็นชื่อตัวแปรหรือฟังก์ชันได้

แนวทางปฏิบัติที่ดีในการตั้งชื่อตัวแปร (Best Practices)

- **ตั้งชื่อให้สื่อความหมาย:**
  - ใช้ชื่อที่บอกถึงหน้าที่หรือเนื้อหาของตัวแปร
  - **ตัวอย่าง:** `counter` แทนที่จะใช้ `c`, `total_sum` แทน `ts`
- **ใช้ snake_case สำหรับชื่อที่มีหลายคำ (นิยมในภาษา Python):**
  - ใช้ขีดล่างเชื่อมคำทั้งหมดเป็นตัวพิมพ์เล็ก
  - **ตัวอย่าง:** `user_name`, `total_amount`

```
def calculate_total_price(product_price, tax_rate):
    total_price = product_price + (product_price * tax_rate)
    return total_price
```

Listing 2.3: ตัวอย่างการใช้ snake_case ใน Python

- **ใช้ PascalCase สำหรับชื่อคลาสหรือชนิดข้อมูล:**
  - ตัวอักษรตัวแรกของทุกคำเป็นพิมพ์ใหญ่ ไม่มีตัวคั่น
  - นิยมใช้ในภาษา C#, Java, Python (ชื่อคลาส)
  - **ตัวอย่าง:** `CustomerAccount`, `TotalAmount`

```python
class CustomerAccount:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance
```

Listing 2.4: ตัวอย่างการใช้ PascalCase สำหรับชื่อคลาส

- **กระชับแต่ชัดเจน:**
  - อย่าตั้งชื่อยาวเกินความจำเป็น
  - **ตัวอย่าง:** `user_id` ดีกว่า `identifier_for_the_user`
- **หลีกเลี่ยงการใช้ตัวอักษรตัวเดียว:**
  - ยกเว้นในกรณีดัชนีวนลูป เช่น `i`, `j`, `k`
  - **ตัวอย่าง:** ใช้ `index` แทน `i`, `count` แทน `c`
- **ใช้รูปแบบชื่อให้สม่ำเสมอ:**
  - หากใช้ `snake_case`, `camelCase` หรือ `PascalCase` ควรใช้ให้เหมือนกันทั้งโปรแกรม
  - ควรเลือกตามมาตรฐานของภาษาโปรแกรมนั้น ๆ

Table 2.3: ตัวอย่างการตั้งชื่อในรูปแบบต่าง ๆ

Purpose	PascalCase	snake_case
Class name	`CustomerAccount`	—
Variable name	—	`customer_account`
Function name	—	`calculate_total()`
Constant name	—	`MAX_VALUE`
File name	—	`user_profile.py`

หมายเหตุ: ตามแนวทาง **PEP 8** ซึ่งเป็นมาตรฐานการเขียนโค้ดของภาษา Python แนะนำให้ใช้ `snake_case` สำหรับชื่อตัวแปรและฟังก์ชัน และใช้ `PascalCase` สำหรับชื่อคลาส เพื่อให้โค้ดอ่านง่ายและมีความสอดคล้องกันในชุมชนผู้ใช้ Python การใช้ `camelCase` แม้จะไม่ผิดทางไวยากรณ์ แต่ถือว่าไม่สอดคล้องกับธรรมเนียมของภาษา Python และควรหลีกเลี่ยงเว้นแต่จำเป็นต้องใช้งานร่วมกับไลบรารีจากภาษาที่ใช้รูปแบบดังกล่าว (เช่น JavaScript)

```python
username = "john_doe"
total_price = 150.75
_is_valid = True
user1 = "Alice"
max_value = 100
```

Listing 2.5: Eตัวอย่างชื่อตัวแปรที่ถูกต้อง

```
1variable = 10         # Starts with a digit
user-name = "Alice"    # Contains a hyphen
total$amount = 100.0   # Contains a special character ($)
class = "Math"         # Reserved keyword
```

Listing 2.6: ตัวอย่างชื่อตัวแปรที่ไม่ถูกต้อง

การปฏิบัติตามกฎและแนวทางเหล่านี้จะช่วยให้โค้ดของคุณอ่านง่าย ดูแลรักษาได้ง่าย และไม่เกิดข้อผิดพลาดเกี่ยวกับชื่อของตัวแปร

2.3 ชนิดของข้อมูล ()

ในภาษา Python ชนิดของข้อมูลที่พบบ่อยครอบคลุมโครงสร้างหลากหลายที่มีความสำคัญต่อการจัดการข้อมูลอย่างมีประสิทธิภาพ ชนิดข้อมูลแบบตัวเลข (Numeric Types) ประกอบด้วยจำนวนเต็ม (Integer) ซึ่งใช้แทนตัวเลขที่ไม่มีทศนิยม, ทศนิยม (Float) สำหรับตัวเลขที่มีจุดทศนิยม, และจำนวนเชิงซ้อน (Complex) ที่ใช้แทนตัวเลขที่มีทั้งส่วนจริงและส่วนจินตภาพ รองรับการคำนวณทางคณิตศาสตร์ในหลายรูปแบบ

ชนิดของลำดับข้อมูล (Sequence Types) เช่น สตริง (String), ลิสต์ (List), และทูเพิล (Tuple) ทำหน้าที่เป็นภาชนะจัดเก็บข้อมูลแบบเรียงลำดับ โดยที่สตริงใช้แทนชุดตัวอักษร, ลิสต์สามารถเก็บหลายค่าและแก้ไขได้ ส่วนทูเพิลเก็บหลายค่าแต่ไม่สามารถเปลี่ยนแปลงได้

ชนิดข้อมูลแบบบูลีน (Boolean) มีบทบาทสำคัญในการดำเนินการทางตรรกะ โดยแทนค่าความจริงในรูปแบบ True หรือ False ซึ่งเป็นพื้นฐานในการตัดสินใจในโปรแกรม นอกจากนี้ ดิกชันนารี (Dictionary) ยังเป็นเครื่องมือสำคัญในการจัดเก็บข้อมูลในรูปแบบคู่คีย์และค่า (key-value pair) ทำให้สามารถเข้าถึงข้อมูลอย่างมีประสิทธิภาพ

ชนิดของข้อมูลเหล่านี้ถือเป็นพื้นฐานสำคัญของการเขียนโปรแกรมด้วย Python ทำให้นักพัฒนาสามารถสร้างโค้ดที่ยืดหยุ่น แข็งแรง และตอบโจทย์การใช้งานในหลายบริบทได้อย่างชัดเจนและมีประสิทธิภาพ

2.3.1 ชนิดข้อมูลที่ใช้บ่อยในภาษา Python

1. **ชนิดตัวเลข (Numeric Types)**

   - **Integer (int):** ตัวเลขจำนวนเต็ม ไม่รวมทศนิยม

   ```
   age = 25
   ```

   - **Float (float):** ตัวเลขที่มีจุดทศนิยม

   ```
   height = 5.6
   ```

   - **Complex (complex):** จำนวนเชิงซ้อน มีทั้งส่วนจริงและส่วนจินตภาพ

   ```
   complex_num = 3 + 4j
   ```

2. **ชนิดลำดับข้อมูล (Sequence Types)**

   - **String (str):** ลำดับของตัวอักษร

   ```
   name = "Alice"
   class_name = 'Computer Programming'
   ```

   - **List (list):** ชุดข้อมูลที่เรียงลำดับ สามารถเปลี่ยนแปลงค่าได้

```
fruits = ["apple", "banana", "cherry"]
mix_list = ['car', 3, "mango", 7.5]
```

- **Tuple (tuple):** ชุดข้อมูลที่เรียงลำดับ แต่ไม่สามารถเปลี่ยนแปลงค่าได้

```
coordinates = (10, 20)
```

3. **ชนิดข้อมูลบูลีน (Boolean Type)**

- **Boolean (bool):** ใช้แทนค่า True หรือ False

```
is_student = True
```

4. **ชนิดเซต (Set Type)**

- **Set (set):** ชุดข้อมูลที่ไม่เรียงลำดับและไม่มีค่าซ้ำ

```
my_set = {1, 2, 3, 4}
another_set = {'apple', 'banana', 'cherry'}
```

5. **ชนิดดิกชันนารี (Dictionary Type)**

- **Dictionary (dict):** เก็บข้อมูลในรูปแบบคู่ key-value

```
student = {"name": "Alice", "age": 25}
```

Table 2.4: ชนิดข้อมูลที่ใช้บ่อยในภาษา Python พร้อมตัวอย่าง

ชนิดข้อมูล (Data Type)	**คำอธิบาย**	**ตัวอย่างโค้ด**
`int` (จำนวนเต็ม)	ตัวเลขที่ไม่มีจุดทศนิยม	`x = 10`
`float` (ทศนิยม)	ตัวเลขที่มีจุดทศนิยม	`pi = 3.14`
`str` (สตริง)	ข้อความหรืออักขระ	`name = "Alice"`
`bool` (ตรรกะ)	ค่า `True` หรือ `False`	`is_valid = True`
`list` (ลิสต์)	ลำดับข้อมูลที่แก้ไขได้	`numbers = [1, 2, 3]`
`tuple` (ทูเพิล)	ลำดับข้อมูลที่แก้ไขไม่ได้	`point = (10, 20)`
`dict` (ดิกชันนารี)	คู่คีย์กับค่า (Key-Value)	`user = {"name": "Bob", "age": 25}`
`set` (เซต)	ชุดข้อมูลที่ไม่ซ้ำกัน	`colors = {"red", "blue", "green"}`

2.3.2 การแปลงชนิดข้อมูล (Type Conversion)

ในภาษา Python ความสามารถในการแปลงชนิดของข้อมูล (Type Conversion) โดยใช้ฟังก์ชันในตัว (built-in functions) ช่วยเพิ่มความยืดหยุ่นในการประมวลผลข้อมูล ฟังก์ชันเหล่านี้ช่วยให้สามารถแปลงข้อมูลจากชนิดหนึ่งไปเป็นอีกชนิดหนึ่งได้อย่างราบรื่น ไม่ว่าจะเป็นการแปลงจากจำนวนเต็มเป็นทศนิยม จากสตริงเป็นจำนวนเต็ม หรือในทางกลับกัน การแปลงเหล่านี้ช่วยให้โค้ดมีความยืดหยุ่นและชัดเจนมากขึ้น ส่งเสริมแนวทางการเขียนโปรแกรมที่มีประสิทธิภาพ และสามารถประยุกต์ใช้กับโจทย์ที่หลากหลายได้อย่างมีประสิทธิผล

```python
# Converting int to float
age = 25
height = float(age)

# Converting float to int
height = 5.6
age = int(height)

# Converting string to int
num_str = "123"
num = int(num_str)
```

Listing 2.7: Example of Python type conversion

สามารถแปลงค่าจากชนิดข้อมูลหนึ่งไปยังอีกชนิดได้ด้วยฟังก์ชันในตัวของ Python ตัวอย่างข้างต้นแสดงการแปลงชนิดข้อมูลในภาษา Python โดยใช้ฟังก์ชันในตัว เริ่มจากกำหนดค่าจำนวนเต็ม 25 ให้กับตัวแปร `age` แล้วแปลงเป็นจำนวนทศนิยมด้วยฟังก์ชัน `float()` จากนั้นแปลงค่าทศนิยม 5.6 เป็นจำนวนเต็มด้วย `int()` ซึ่งจะได้ผลลัพธ์เป็น 5 เนื่องจากทศนิยมจะถูกตัดออก และสุดท้ายแปลงสตริง "123" เป็นจำนวนเต็มด้วย `int()` ได้ค่าผลลัพธ์เป็น 123 แสดงให้เห็นถึงความสามารถของ Python ในการปรับเปลี่ยนรูปแบบข้อมูลได้อย่างยืดหยุ่นภายในโปรแกรม

2.4 ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

ภาษา Python เป็นภาษาการเขียนโปรแกรมที่มีความยืดหยุ่นและทรงพลัง โดยมีตัวดำเนินการทางคณิตศาสตร์หลากหลายที่ใช้ในการดำเนินการคำนวณทั้งระดับพื้นฐานและขั้นสูง การเข้าใจตัวดำเนินการเหล่านี้ถือเป็นสิ่งสำคัญสำหรับผู้ที่ใช้ Python ไม่ว่าจะเพื่อการคำนวณง่าย ๆ หรือการวิเคราะห์ข้อมูลที่ซับซ้อน ตัวดำเนินการหลักใน Python ได้แก่ การบวก การลบ การคูณ การหาร การหารเอาเศษ การยกกำลัง และการหารปัดเศษลง

$$y = x^2 + 2x + 1$$

$y, x, 1$	คือ ตัวถูกดำเนินการ (Operand)
$=, +,$ exponent	คือ ตัวดำเนินการ (Operator)
$y = x^2 + 2x + 1$	คือ นิพจน์ (Expression)

Table 2.5: การเปรียบเทียบตัวดำเนินการทางคณิตศาสตร์

Operator	ความหมาย	ตัวอย่าง
+	การบวกค่าของสองตัวแปร หรือบวกค่าบวก	$x + y$, +2
-	การลบค่าของตัวแปรด้านขวาออกจากด้านซ้าย หรือค่าลบ	$x - y$, -2
*	การคูณตัวแปรสองตัว	$x * y$
/	การ หาร ตัวแปร ด้าน ซ้าย ด้วย ด้าน ขวา (ผลลัพธ์เป็นทศนิยมเสมอ)	x/y
%	การหารเอาเศษ (modulus)	$x\%y$
//	การหารแบบปัดเศษลง (floor division)	$x//y$
**	การยกกำลัง (exponentiation)	$x**y$

1. การบวก (+) ใช้สำหรับบวกค่าตัวเลข หรือใช้สำหรับเชื่อมข้อความ (concatenate) ได้ในบางกรณี

```python
# Addition of integers
a = 10
b = 20
result = a + b
print(result)  # Output: 30

# Addition of floats
c = 1.5
d = 2.5
result = c + d
print(result)  # Output: 4.0

# Concatenation of strings
str1 = "Hello"
str2 = " World"
result = str1 + str2
print(result)  # Output: Hello World
```

Listing 2.8: การบวกจำนวนเต็ม ทศนิยม และสตริง

2. การลบ (-) ใช้ในการลบค่าของตัวแปรด้านขวาออกจากด้านซ้าย

```python
# Subtraction of integers
a = 10
b = 5
result = a - b
print(result)  # Output: 5

# Subtraction of floats
c = 5.5
d = 2.0
result = c - d
print(result)  # Output: 3.5
```

Listing 2.9: การลบจำนวนเต็มและทศนิยม

3. การคูณ (*) ใช้คูณตัวเลข หรือใช้ทำซ้ำสตริง

```python
# Multiplication of integers
a = 5
b = 4
result = a * b
print(result)  # Output: 20

# Multiplication of floats
c = 2.5
d = 4.0
result = c * d
print(result)  # Output: 10.0

# Repetition of strings
str1 = "Hi"
result = str1 * 3
print(result)  # Output: HiHiHi
```

Listing 2.10: การคูณจำนวนเต็ม ทศนิยม และสตริง

4. การหาร (/) หาร่าของตัวแปรแรกด้วยตัวแปรที่สอง ผลลัพธ์เป็นค่าทศนิยมเสมอ

```python
# Division of integers
a = 20
b = 4
result = a / b
print(result)  # Output: 5.0

# Division of floats
c = 7.5
d = 2.5
result = c / d
print(result)  # Output: 3.0
```

Listing 2.11: การหารจำนวนเต็มและทศนิยม

5. การหารแบบปัดเศษลง (//) เป็นการหารที่ผลลัพธ์ปัดเศษลงเป็นจำนวนเต็ม

```
# Floor division of integers
a = 20
b = 3
result = a // b
print(result)  # Output: 6

# Floor division of floats
c = 7.5
d = 2.5
result = c // d
print(result)  # Output: 3.0
```

Listing 2.12: การหารแบบปัดเศษลงของจำนวนเต็มและทศนิยม

6. การหารเอาเศษ (%) ใช้หาเศษจากการหาร ช่วยในการตรวจสอบเลขคี่หรือเลขคู่

```
# Modulus of integers
a = 10
b = 3
result = a % b
print(result)  # Output: 1

# Modulus of floats
c = 7.5
d = 2.5
result = c % d
print(result)  # Output: 0.0
```

Listing 2.13: การหารเอาเศษของจำนวนเต็มและทศนิยม

7. การยกกำลัง (**) ยกค่าตัวแปรแรกเป็นกำลังของตัวแปรที่สอง

```
# Exponentiation of integers
a = 2
b = 3
result = a ** b
print(result)  # Output: 8

# Exponentiation of floats
c = 3.5
d = 2
result = c ** d
print(result)  # Output: 12.25
```

Listing 2.14: การยกกำลังของจำนวนเต็มและทศนิยม

ลำดับความสำคัญของตัวดำเนินการ (Operator Precedence) ลำดับความสำคัญของตัวดำเนินการคือกฎที่กำหนดว่าการดำเนินการใดควรถูกประเมินก่อนเมื่อมีหลายตัวดำเนินการในนิพจน์ ตัวอย่างเช่น การคูณและการหารมีลำดับสูงกว่าการบวกและการลบ เราสามารถใช้วงเล็บเพื่อควบคุมลำดับการดำเนินการให้เป็นไปตามที่ต้องการ

Table 2.6: ลำดับความสำคัญของตัวดำเนินการในภาษา Python

ตัวดำเนินการ
()
**
*, /, %, //
+, -
<=, <, >, >=
==, !=
=

$$5 + 18 \div 2 - 25\%4 = ?$$

ตัวอย่างการใช้งานจริง ตัวดำเนินการทางคณิตศาสตร์มีประโยชน์ในหลากหลายสถานการณ์ เช่น ในการคำนวณทางการเงิน การหากำไร ขาดทุน ดอกเบี้ย หรือในงานวิทยาศาสตร์ที่ต้องใช้การยกกำลัง และการใช้ modulus เพื่อตรวจสอบรอบเวลา ปีอธิกสุรทิน และการเขียนโปรแกรมที่ต้องมีการวนรอบ

บทสรุป การเข้าใจและใช้งานตัวดำเนินการทางคณิตศาสตร์ของ Python ได้อย่างถูกต้องถือเป็นรากฐานสำคัญของการเขียนโปรแกรม ไม่เพียงช่วยในการคำนวณพื้นฐานเท่านั้น แต่ยังนำไปสู่การสร้างโปรแกรมที่ซับซ้อนได้อย่างมีประสิทธิภาพและแม่นยำ

2.5 การรับข้อมูลจากผู้ใช้ (User Input)

การรับข้อมูลจากผู้ใช้ถือเป็นส่วนสำคัญของการเขียนโปรแกรม เพราะช่วยให้ผู้ใช้งานสามารถโต้ตอบกับซอฟต์แวร์ได้ ในภาษา Python ฟังก์ชัน `input()` ใช้ในการรับค่าที่ผู้ใช้ป้อนผ่านหน้าจอคอนโซล โดยผู้ใช้จะป้อนข้อมูลหรือคำสั่งที่มีผลต่อพฤติกรรมของโปรแกรม เมื่อมีข้อความแจ้งเตือน โปรแกรมจะรอให้ผู้ใช้พิมพ์ค่าลงไป ซึ่งอาจเป็นข้อความธรรมดาหรือข้อมูลที่ซับซ้อน การจัดการกับข้อมูลที่รับเข้ามาอย่างระมัดระวังจึงเป็นสิ่งสำคัญในการรับประกันว่าโปรแกรมจะทำงานอย่างถูกต้องและใช้งานง่าย การเลือกของผู้ใช้สามารถเปลี่ยนแปลงทิศทางการทำงานของโปรแกรมได้ ไม่ว่าจะเป็นในโปรแกรมพื้นฐานหรือแอปพลิเคชันที่มีอินเทอร์เฟซแบบกราฟิก การรับข้อมูลจากผู้ใช้จึงเป็นกลไกสำคัญในการสร้างประสบการณ์ใช้งานที่ตอบสนองต่อผู้ใช้ โปรแกรมเมอร์ที่เชี่ยวชาญในการจัดการอินพุตจะสามารถพัฒนาแอปพลิเคชันที่ตอบโจทย์และใช้งานได้หลากหลายรูปแบบ

2.5.1 การรับข้อมูลจากผู้ใช้ด้วย input()

ในภาษา Python ฟังก์ชัน `input()` ช่วยให้โปรแกรมสามารถโต้ตอบกับผู้ใช้ได้โดยแสดงข้อความแจ้งให้ผู้ใช้กรอกข้อมูล ซึ่งค่าที่รับเข้ามาจะถูกจัดเก็บในรูปแบบข้อความ (string) เสมอ ไม่ว่าจะเป็นตัวเลขหรือข้อความธรรมดา แม้

จะเป็นข้อความที่ดูเหมือนตัวเลขก็ตาม ผู้พัฒนาจึงต้องทำการแปลงประเภทข้อมูล (type conversion) หากต้องการนำไปใช้ในทางคำนวณ ฟังก์ชันนี้จึงเป็นจุดเริ่มต้นของการสร้างโปรแกรมแบบโต้ตอบกับผู้ใช้ การแปลงและตรวจสอบความถูกต้องของข้อมูลเป็นสิ่งสำคัญที่จะทำให้โปรแกรมมีประสิทธิภาพและใช้งานได้จริง

```
# Getting user input
name = input("Enter your name: ")
age = input("Enter your age: ")
print("Hello, " + name + "! You are " + age + " years old.")
```

Listing 2.15: ตัวอย่างการรับข้อมูลจากผู้ใช้

2.5.2 การแปลงประเภทข้อมูลที่รับเข้ามา

เนื่องจาก `input()` จะคืนค่าข้อมูลเป็นข้อความ (string) เสมอ โปรแกรมเมอร์จึงต้องทำการแปลงข้อมูลให้เป็นประเภทที่เหมาะสม เช่น แปลงเป็นตัวเลขจำนวนเต็มหรือทศนิยม เพื่อใช้ในการคำนวณหรือตรวจสอบค่าที่ผู้ใช้ป้อนเข้ามา การแปลงประเภทข้อมูลจึงเป็นสิ่งสำคัญในการประมวลผลข้อมูลให้ตรงกับความต้องการของโปรแกรม ช่วยให้โปรแกรมสามารถจัดการข้อมูลจากผู้ใช้ได้อย่างถูกต้องและเชื่อถือได้

```
# Getting user input and converting to int
age = input("Enter your age: ")
age = int(age)

# Getting user input and converting to float
height = input("Enter your height: ")
height = float(height)

print("You are " + str(age) + " years old and " + str(height) + " feet tall
    .")
```

Listing 2.16: ตัวอย่างการแปลงข้อมูลจากผู้ใช้

2.6 ตัวอย่างการประยุกต์ใช้งานจริง (Practical Example)

การผสานการใช้ตัวแปร ประเภทข้อมูล และการรับค่าจากผู้ใช้เข้าด้วยกัน เป็นพื้นฐานสำคัญของการเขียนโปรแกรมที่มีการโต้ตอบและตอบสนองต่อข้อมูลที่เปลี่ยนแปลงได้อย่างยืดหยุ่น

2.6.1 ตัวอย่างที่ 1: โปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้า

โปรแกรมนี้รับข้อมูลจากผู้ใช้ ได้แก่ อายุและส่วนสูง แล้วแปลงข้อมูลให้เหมาะสมกับประเภทที่ต้องการใช้งาน โดยใช้ฟังก์ชัน `input()` เพื่อรับข้อมูล และใช้ `int()` และ `float()` สำหรับแปลงค่าที่ป้อนเข้ามา ก่อนนำไปแสดงผลบนหน้าจอ

Algorithm 2: Algorithm to Calculate Area of a Rectangle

Input: Width and height of the rectangle
Output: The area value

```
1 begin
2     Print "Enter the width of the rectangle:";
3     Read width;
4     Print "Enter the height of the rectangle:";
5     Read height;
6     area = width * height;
7     Print "The area of the rectangle is", area;
```

```python
# Getting user input and converting to float
width = input("Enter the width of the rectangle: ")
width = float(width)

height = input("Enter the height of the rectangle: ")
height = float(height)

# Calculating area
area = width * height

# Displaying result
print("The area of the rectangle is " + str(area))
```

Listing 2.17: โปรแกรมคำนวณค่าพื้นที่ของสี่เหลี่ยม

2.6.2 ตัวอย่างที่ 2: โปรแกรมคำนวณค่าดัชนีมวลกาย (BMI)

โปรแกรมนี้ใช้คำนวณค่าดัชนีมวลกาย (BMI) จากน้ำหนักและส่วนสูงที่ผู้ใช้ป้อนเข้ามา

Algorithm 3: อัลกอริทึมสำหรับคำนวณ BMI

Input: น้ำหนักและส่วนสูงของผู้ใช้
Output: ค่าดัชนีมวลกาย

```
1 begin
2     Print "Enter your weight in kilograms:";
3     Read weight;
4     Print "Enter your height in meters:";
5     Read height;
6     bmi = weight / (height * height);
7     Print "Your BMI is", bmi;
```

```python
# Program to calculate BMI

# Getting user input for weight and height
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))

# Calculating BMI
bmi = weight / (height * height)

# Displaying the result
print("Your BMI is", bmi)
```

Listing 2.18: โปรแกรมคำนวณ BMI

คำอธิบาย:

- โปรแกรมจะแจ้งให้ผู้ใช้กรอกน้ำหนักและส่วนสูง
- ข้อมูลที่รับเข้ามาจะถูกแปลงเป็นเลขทศนิยม
- ค่าดัชนีมวลกาย (BMI) คำนวณจากสูตร: `BMI = weight / (height * height)`
- แสดงผลลัพธ์บนหน้าจอ

2.6.3 ตัวอย่างที่ 3: โปรแกรมแปลงอุณหภูมิ

โปรแกรมนี้ใช้แปลงอุณหภูมิจากองศาเซลเซียสเป็นฟาเรนไฮต์

Algorithm 4: อัลกอริทึมแปลงอุณหภูมิจาก Celsius เป็น Fahrenheit

Input: อุณหภูมิในหน่วยเซลเซียส
Output: อุณหภูมิในหน่วยฟาเรนไฮต์

```
1 begin
2     Print "Enter temperature in Celsius:";
3     Read celsius;
4     fahrenheit = (celsius * 9/5) + 32;
5     Print "Temperature in Fahrenheit is", fahrenheit;
```

```python
# Program to convert Celsius to Fahrenheit

# Getting user input for temperature in Celsius
celsius = float(input("Enter temperature in Celsius: "))

# Converting Celsius to Fahrenheit
fahrenheit = (celsius * 9/5) + 32

# Displaying the result
print("Temperature in Fahrenheit is", fahrenheit)
```

Listing 2.19: โปรแกรมแปลงอุณหภูมิ Celsius เป็น Fahrenheit

คำอธิบาย:

- โปรแกรมจะรับค่าอุณหภูมิจากผู้ใช้
- แปลงค่าเป็น float เพื่อให้รองรับเลขทศนิยม
- แปลงค่าเซลเซียสเป็นฟาเรนไฮต์ด้วยสูตร: `Fahrenheit = (Celsius * 9/5) + 32`
- แสดงผลลัพธ์บนหน้าจอ

ตัวอย่างทั้งหมดนี้แสดงให้เห็นถึงการใช้งานตัวแปร ประเภทข้อมูล และการรับค่าจากผู้ใช้ในทางปฏิบัติ เพื่อเสริมสร้างความเข้าใจในแนวคิดพื้นฐานของการเขียนโปรแกรมภาษา Python อย่างชัดเจน

บทที่ 2 โจทย์และแบบฝึกหัด: ตัวแปร ชนิดข้อมูล และการรับข้อมูลจากผู้ใช้

2.1 การตั้งชื่อตัวแปรในภาษา Python
อธิบายกฎและแนวปฏิบัติที่ดีในการตั้งชื่อตัวแปร พร้อมยกตัวอย่างชื่อที่ถูกและผิด

2.2 ประเภทข้อมูลเชิงตัวเลข (Numeric Types)
อธิบายความแตกต่างระหว่าง int, float และ complex พร้อมยกตัวอย่างการใช้งาน

2.3 ข้อมูลแบบลำดับ (Sequence Types)
เปรียบเทียบ string, list และ tuple และยกตัวอย่างกรณีการใช้งานที่แตกต่างกัน

2.4 ข้อมูลแบบตรรกะ (Boolean)
อธิบายวัตถุประสงค์ของ Boolean และยกตัวอย่างนิพจน์ทางตรรกะใน Python

2.5 การแปลงประเภทข้อมูล (Type Conversion)
อธิบายความสำคัญของการแปลงประเภทข้อมูล และแสดงตัวอย่างการแปลงด้วยฟังก์ชันในตัว

2.6 การรับข้อมูลจากผู้ใช้ (User Input)
อธิบายการทำงานของฟังก์ชัน input() และการแปลงข้อมูลจาก string เป็นประเภทอื่น

2.7 ตัวอย่างการใช้งานจริง: เครื่องคิดเลข
*เขียนโปรแกรม Python ที่รับตัวเลข 2 ค่า และตัวดำเนินการ + - * / แล้วแสดงผลลัพธ์*

2.8 การจัดการข้อผิดพลาดในการป้อนข้อมูล
อธิบายการจัดการข้อผิดพลาดในการรับค่าจากผู้ใช้ พร้อมตัวอย่างการจัดการประเภทข้อมูลผิดหรือหารด้วยศูนย์

2.9 ความเข้าใจการแปลงข้อมูล
อธิบายความสำคัญของการแปลงประเภทข้อมูลในบทนี้ พร้อมยกตัวอย่างการแปลงที่จำเป็น

2.10 การตรวจสอบความถูกต้องของข้อมูล
เขียนโปรแกรมที่รับจำนวนเต็มบวกจากผู้ใช้ หากไม่ถูกให้ป้อนใหม่ และแสดงค่ารากที่สอง

ภาคผนวก

A2.1 แหล่งเรียนรู้เพิ่มเติม

หนังสือ:

- *Python Programming: An Introduction to Computer Science* โดย John Zelle
- *Python Cookbook* โดย David Beazley และ Brian K. Jones

บทเรียนออนไลน์:

- Python Variables - W3Schools
- Data Types in Python - GeeksforGeeks

คอร์สเรียน:

- Coursera: Python Data Structures – มหาวิทยาลัยมิชิแกน
- Codecademy: Learn Python 3

A2.2 เอกสารอ้างอิง

- Python Software Foundation. (2024). เอกสาร Python – ข้อมูลประเภทต่าง ๆ: https://docs.python.org/3/library/datatypes.html
- Lutz, M. (2013). *Learning Python* (ฉบับที่ 5). O'Reilly Media.

A2.3 แบบฝึกหัดเพิ่มเติม

แบบฝึกหัดที่ 1: การประกาศและกำหนดค่าตัวแปร

- ประกาศตัวแปร 'name' และกำหนดชื่อของคุณ แล้วพิมพ์ค่าตัวแปร
- สร้างตัวแปร 'a' และ 'b' แล้วสลับค่าระหว่างกัน

แบบฝึกหัดที่ 2: ประเภทข้อมูล

- เขียนโปรแกรมให้ผู้ใช้ป้อนอายุ แล้วแจ้งว่าเป็นเยาวชน ผู้ใหญ่ หรือผู้สูงอายุ
- สร้าง list ของผลไม้ที่ชอบ แล้วแสดงผลรายการ

แบบฝึกหัดที่ 3: การรับข้อมูลจากผู้ใช้

- เขียนโปรแกรมให้ผู้ใช้ป้อนปีเกิด แล้วคำนวณอายุ
- เขียนสคริปต์ที่รับค่าตัวเลข 2 ค่า แล้วแสดงผลคูณ

A2.4 ตัวอย่างเชิงปฏิบัติ

ตัวอย่างที่ 1: การรับข้อมูลและแปลงประเภท

```python
# Function to calculate travel time
def calculate_travel_time(distance, speed):
    return distance / speed

# Get user input for distance and speed
distance_km = float(input("Enter distance to travel (in kilometers): "))
speed_kmh = float(input("Enter speed (in km/h): "))

# Calculate travel time
time_hours = calculate_travel_time(distance_km, speed_kmh)

# Display result
print(f"Estimated travel time: {time_hours:.2f} hours")
```

Listing 2.20: โปรแกรมคำนวณเวลาในการเดินทาง

ตัวอย่างที่ 2: การทำงานกับ list และ dictionary

```python
# Python program to demonstrate working with lists and dictionaries
# List of favorite movies
favorite_movies = ["Inception", "The Matrix", "Interstellar"]

# Print the list
print("Favorite Movies:", favorite_movies)

# Dictionary of movie ratings
movie_ratings = {
    "Inception": 8.8,
    "The Matrix": 8.7,
    "Interstellar": 8.6
}

# Print the dictionary
print("Movie Ratings:", movie_ratings)

# Adding a new movie rating
movie_ratings["The Godfather"] = 9.2
print("Updated Movie Ratings:", movie_ratings)
```

Listing 2.21: การใช้ list และ dictionary กับข้อมูลภาพยนตร์

ตัวอย่างที่ 3: คำนวณพื้นที่ของรูปห้าเหลี่ยมด้านเท่า

```python
import math

# Function to calculate area of a regular pentagon using only side length
def calculate_pentagon_area(side_length):
    area = (5 / 4) * (side_length ** 2) * (1 / math.tan(math.pi / 5))
    return area

# Get user input
side = float(input("Enter the length of a side of the pentagon: "))

# Calculate and display area
area = calculate_pentagon_area(side)
print(f"The area of the regular pentagon is: {area:.2f}")
```

Listing 2.22: คำนวณพื้นที่ของรูปห้าเหลี่ยมด้านเท่า

พื้นที่ของรูปห้าเหลี่ยมด้านเท่าสามารถคำนวณได้โดยใช้สูตร:

$$\text{Area} = \frac{5 \times s \times a}{2} \tag{2.1}$$

โดยที่:

- s คือ ความยาวด้านของห้าเหลี่ยม
- a คือ ความยาว **apothem** (เส้นตรงจากจุดศูนย์กลางของรูปถึงกึ่งกลางของด้าน)

  สูตรนี้ได้จากการแบ่งรูปห้าเหลี่ยมออกเป็น 5 รูปสามเหลี่ยมหน้าจั่ว แล้วหาพื้นที่ของแต่ละสามเหลี่ยมด้วยสูตร $\frac{1}{2} \times \ \times$ ซึ่งในกรณีนี้ฐานคือ s และสูงคือ a แล้วคูณด้วย 5:

$$\text{Area} = 5 \times \left(\frac{1}{2} \times s \times a\right) = \frac{5sa}{2}$$

สูตรนี้เหมาะสำหรับการคำนวณพื้นที่ของรูปห้าเหลี่ยมที่มีความยาวด้านเท่ากันและรู้ค่า apothem แล้วเท่านั้น หากไม่มีค่า apothem อาจต้องใช้สูตรที่ซับซ้อนขึ้นซึ่งเกี่ยวข้องกับมุมภายในและฟังก์ชันตรีโกณมิติ เช่น:

$$\text{Area} = \frac{5}{4} \cdot s^2 \cdot \cot\left(\frac{\pi}{5}\right) \tag{2.2}$$

- $\cot\left(\frac{\pi}{5}\right) \approx 1.37638$
- สูตรนี้ใช้เฉพาะในกรณีที่ทราบเพียงความยาวด้าน s เท่านั้น

บทที่ 3

เงื่อนไขและการควบคุมลำดับการทำงาน

บทนี้นำเสนอภาพรวมของคำสั่งเงื่อนไขและการควบคุมลำดับการทำงานในภาษา Python อย่างละเอียด ครอบคลุมคำสั่ง `if`, `elif`, และ `else` ตลอดจนตัวดำเนินการเปรียบเทียบ (comparison operators) และตัวดำเนินการเชิงตรรกะ (logical operators) พร้อมตัวอย่างการใช้งานจริงเพื่อให้เข้าใจแนวคิดดังกล่าวได้อย่างเป็นรูปธรรม การเข้าใจแนวคิดพื้นฐานเหล่านี้จะช่วยให้เขียนโปรแกรมได้มีประสิทธิภาพและมีตรรกะมากขึ้น โดยสามารถตัดสินใจและเลือกดำเนินการตามเงื่อนไขที่กำหนดได้ คำสั่งเงื่อนไข เช่น `if`, `elif`, และ `else` ช่วยให้สามารถกำหนดให้โปรแกรมดำเนินการตามเงื่อนไขที่ตรงกัน ซึ่งช่วยเพิ่มความยืดหยุ่นในการควบคุมกระบวนการทำงาน และสามารถรองรับสถานการณ์และข้อมูลนำเข้าได้หลากหลาย นอกจากนี้ การเข้าใจตัวดำเนินการเปรียบเทียบมีความสำคัญเนื่องจากช่วยให้สามารถเปรียบเทียบค่าและตัดสินใจตามความสัมพันธ์ของข้อมูลได้อย่างถูกต้อง อีกทั้งตัวดำเนินการเชิงตรรกะยังช่วยให้สามารถรวมหลายเงื่อนไขเข้าด้วยกัน เพื่อสร้างตรรกะที่ซับซ้อนและแม่นยำมากยิ่งขึ้น

ตัวอย่างการใช้งานในบทนี้แสดงให้เห็นถึงโครงสร้างทางไวยากรณ์และการทำงานของคำสั่งเงื่อนไข พร้อมทั้งแสดงให้เห็นว่าคำสั่งเหล่านี้สามารถนำไปประยุกต์ใช้ในสถานการณ์จริงได้อย่างไร วิธีการเรียนรู้ผ่านการปฏิบัติจริงนี้จะช่วยให้เข้าใจเนื้อหาได้ชัดเจนขึ้น และสร้างความมั่นใจในการนำไปใช้ในการเขียนโปรแกรมด้วย Python ได้อย่างถูกต้อง

3.1 คำสั่งเงื่อนไข (Conditional Statements)

คำสั่งเงื่อนไขและการควบคุมลำดับการทำงานในโปรแกรม คือกลไกที่ช่วยให้โปรแกรมสามารถตัดสินใจและดำเนินการตามเงื่อนไขต่าง ๆ ได้ ซึ่งจะใช้คำสั่งเช่น `if`, `elif` และ `else` เพื่อตรวจสอบว่าเงื่อนไขเป็นจริงหรือเท็จ ตัวอย่างเช่น คำสั่ง `if` จะทำงานก็ต่อเมื่อเงื่อนไขเป็นจริง ส่วน `elif` จะตรวจสอบเงื่อนไขอื่น ๆ หากเงื่อนไขก่อนหน้านั้นไม่เป็นจริง และ `else` จะทำงานเมื่อไม่มีเงื่อนไขใด ๆ เป็นจริงเลย โครงสร้างเหล่านี้ทำให้โปรแกรมสามารถตอบสนองต่อข้อมูลนำเข้าและสถานการณ์ที่แตกต่างกันได้อย่างยืดหยุ่นและโต้ตอบได้

ด้วยการใช้คำสั่งเงื่อนไข เราสามารถควบคุมให้โปรแกรมทำงานแตกต่างกันไปตามผลลัพธ์ของเงื่อนไขที่ตรวจสอบ ซึ่งเป็นสิ่งสำคัญในการสร้างโปรแกรมที่มีการโต้ตอบและมีตรรกะที่ซับซ้อน หากไม่มีคำสั่งเงื่อนไข โปรแกรมจะไม่สามารถเลือกทางเลือกใด ๆ ได้เลย ซึ่งจะจำกัดความสามารถและความยืดหยุ่นของโปรแกรมอย่างมาก

การเข้าใจคำสั่งเงื่อนไขจึงเป็นพื้นฐานสำคัญของการเขียนโปรแกรมที่มีประสิทธิภาพและสามารถจัดการสถานการณ์ที่หลากหลายได้อย่างเหมาะสม

แนวคิดสำคัญเกี่ยวกับคำสั่งเงื่อนไข:

- **If Statement:** ทำงานเมื่อเงื่อนไขที่กำหนดเป็นจริง
- **Elif Statement:** ตรวจสอบเงื่อนไขเพิ่มเติม หากเงื่อนไขก่อนหน้าไม่เป็นจริง
- **Else Statement:** ทำงานเมื่อไม่มีเงื่อนไขใด ๆ ที่เป็นจริง
- **ไวยากรณ์ (Syntax):** การจัดวางและย่อหน้าที่ถูกต้องช่วยให้โค้ดอ่านง่ายและทำงานได้ถูกต้อง
- **เงื่อนไขเดียว:** `if condition:`
- **หลายเงื่อนไข:** `if condition1: elif condition2: else:`
- **การประเมินค่าทางบูลีน:** เงื่อนไขจะถูกประเมินเป็น `True` หรือ `False`
- **คำสั่งซ้อนกัน:** สามารถเขียนคำสั่งเงื่อนไขซ้อนกันได้เพื่อสร้างตรรกะที่ซับซ้อนมากขึ้น
- **ควบคุมลำดับการทำงาน:** กำหนดทิศทางการทำงานของโปรแกรมตามผลของเงื่อนไข
- **การทำงานค่าเริ่มต้น:** คำสั่ง `else` ใช้สำหรับกรณีที่ไม่มีเงื่อนไขใดตรงเลย
- **ความชัดเจน:** ทำให้โค้ดอ่านง่าย เข้าใจง่าย และดูแลรักษาง่าย

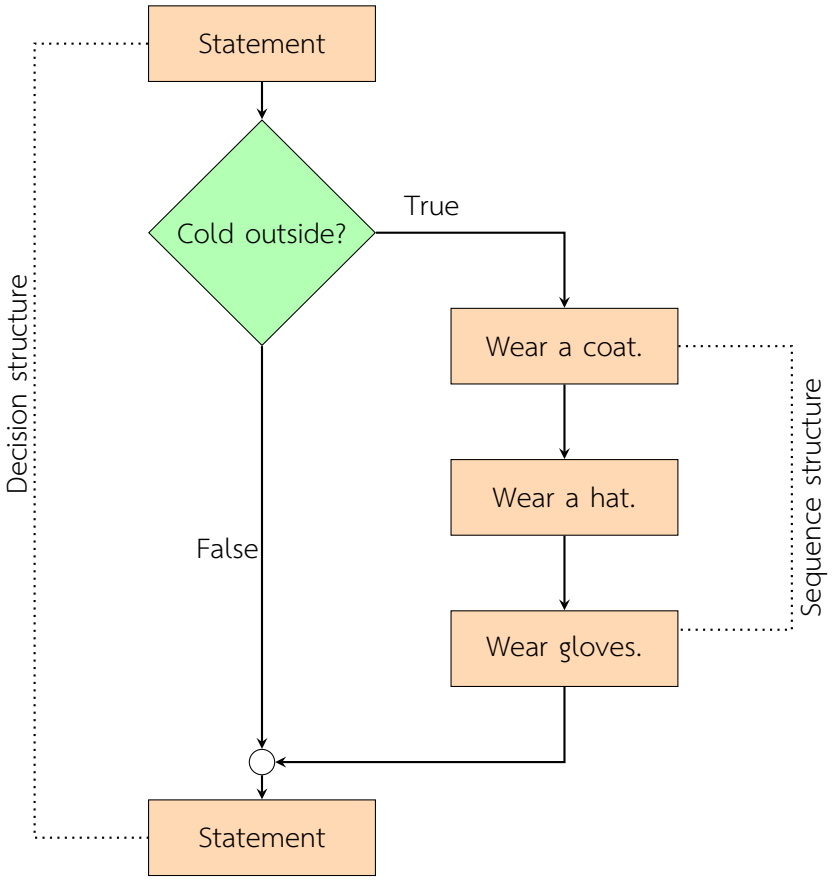

Figure 3.1: Flowchart of Decision Structure

แผนผังนี้แสดงให้เห็นถึงแนวคิดของการควบคุมลำดับการทำงานในโปรแกรม โดยใช้โครงสร้างการตัดสินใจร่วมกับลำดับคำสั่ง โครงสร้างการตัดสินใจซึ่งแสดงด้วยสัญลักษณ์รูปเพชร ทำหน้าที่ตรวจสอบเงื่อนไข เช่น "อากาศข้างนอกหนาวหรือไม่?" หากเงื่อนไขเป็นจริง โปรแกรมจะเข้าสู่โครงสร้างลำดับคำสั่งที่ประกอบด้วย "ใส่เสื้อโค้ต", "ใส่หมวก", และ "ใส่ถุงมือ" ตามลำดับ หากเงื่อนไขเป็นเท็จ โปรแกรมจะข้ามลำดับคำสั่งนี้และดำเนินต่อไปในส่วนถัดไป แผนผังนี้จึงแสดงให้เห็นว่าการตัดสินใจสามารถกำหนดทิศทางของโปรแกรมได้อย่างมีประสิทธิภาพ โดยให้โปรแกรมดำเนินการเฉพาะเมื่อเงื่อนไขที่กำหนดเป็นจริง ซึ่งช่วยเพิ่มความยืดหยุ่นและการตอบสนองของโปรแกรม

3.1.1 คำสั่ง If

คำสั่ง if ช่วยให้สามารถดำเนินการโค้ดบล็อกหนึ่งได้เมื่อเงื่อนไขเป็นจริง ซึ่งเป็นโครงสร้างควบคุมพื้นฐานในโปรแกรมมิ่ง เมื่อเงื่อนไขที่ระบุในคำสั่ง if ถูกประเมินว่าเป็นจริง โปรแกรมจะดำเนินการบล็อกโค้ดที่อยู่ภายใต้ if ที่เยื้องไว้ วิธีนี้ทำให้โปรแกรมสามารถตัดสินใจและดำเนินการตามข้อมูลที่เปลี่ยนแปลงได้ เช่น ตรวจสอบว่าอายุของผู้ใช้เกินเกณฑ์หรือไม่เพื่อให้เข้าถึงฟีเจอร์บางอย่าง หรือว่าตัวแปรตรงตามเงื่อนไขก่อนจะคำนวณต่อไป การดำเนินการแบบมีเงื่อนไขเป็นสิ่งสำคัญสำหรับการสร้างโปรแกรมที่ตอบสนองต่อสถานการณ์และอินพุตที่หลากหลาย หากไม่มีคำสั่ง if โปรแกรมจะเป็นเส้นตรงและไม่สามารถจัดการกับเงื่อนไขต่าง ๆ ได้ ทำให้ขาดความยืดหยุ่นและประสิทธิภาพ

```
1 if condition:
2     # code to execute if the condition is true
```

Listing 3.1: ไวยากรณ์ของ IF

```
1 age = int(input("Please input age: "))
2 if age >= 18:
3     print("You are an adult.")
```

Listing 3.2: ตัวอย่างของ IF

Algorithm 5: อัลกอริทึมเพื่อตรวจสอบว่าบุคคลเป็นผู้ใหญ่หรือไม่

Input: อายุ

Output: ข้อความแจ้งว่าบุคคลนั้นเป็นผู้ใหญ่หรือไม่

```
1 begin
2     แสดง "Please input age:";
3     อ่าน age;
4     if age >= 18 then
5         แสดง "You are an adult.";
```

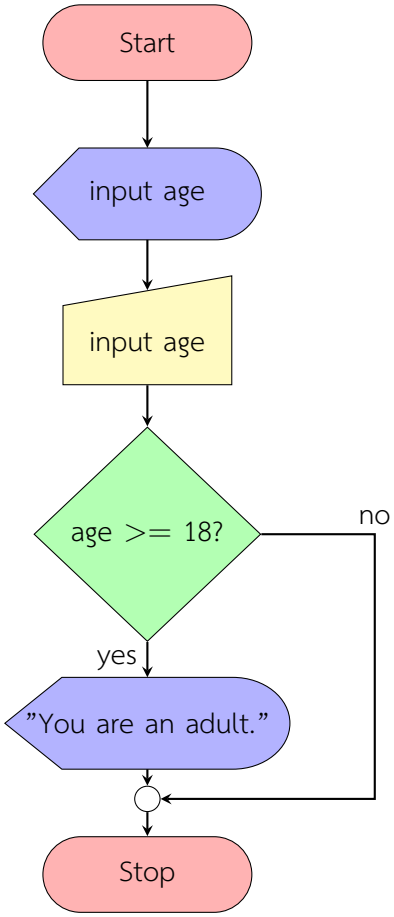

Figure 3.2: ผังงานสำหรับโปรแกรมตรวจสอบอายุ

Algorithm 6: แบบฝึกหัด: อัลกอริทึมสำหรับคำนวณค่าเฉลี่ยของคะแนนสอบ

Input: คะแนนสอบ 3 วิชา
Output: ค่าเฉลี่ย และ "Congratulations!" หากเกิน 95

```
1 begin
2     รับคะแนนสอบที่หนึ่ง;
3     รับคะแนนสอบที่สอง;
4     รับคะแนนสอบที่สาม;
5     คำนวณค่าเฉลี่ย;
6     แสดงผลค่าเฉลี่ย;
7     if ค่าเฉลี่ย > 95 then
8         แสดง "Congratulations!";
```

3.1.2 คำสั่ง Elif

คำสั่ง elif (ย่อมาจาก "else if") ช่วยให้สามารถตรวจสอบหลายเงื่อนไขได้ โดยสามารถใช้หลังจาก if เพื่อประเมินเงื่อนไขเพิ่มเติมหากเงื่อนไขก่อนหน้าไม่เป็นจริง ตัวอย่างเช่นในระบบให้เกรด นักเรียนอาจได้ "A" ถ้าคะแนนมากกว่า 90 หรือ "B" ถ้าอยู่ในช่วง 80 ถึง 89 เป็นต้น คำสั่ง elif ช่วยให้เขียนโปรแกรมที่มีหลายเงื่อนไขอย่างชัดเจน และหากไม่มีเงื่อนไขใดเป็นจริงเลย ก็สามารถใช้ else เพื่อกำหนดผลลัพธ์เริ่มต้นได้ วิธีนี้ช่วยให้โค้ดอ่านง่าย มีโครงสร้าง และควบคุมทิศทางของโปรแกรมอย่างชัดเจน

ไวยากรณ์ของ Elif:

```
if condition1:
    # code to execute if condition1 is true
elif condition2:
    # code to execute if condition2 is true
```

Listing 3.3: Syntax of Elif condition

ตัวอย่างที่ 1:

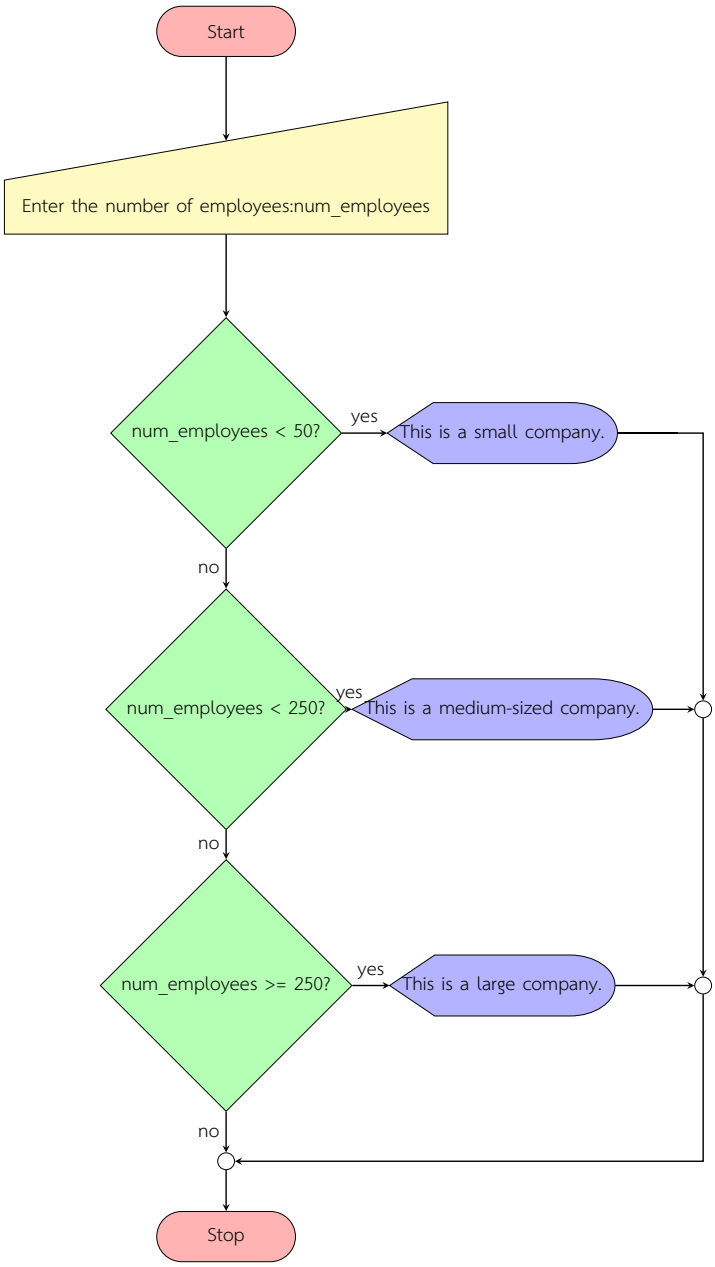

Figure 3.3: ผังงานสำหรับการกำหนดขนาดของบริษัท

```python
num_employees = int(input("Enter the number of employees: "))
if num_employees < 50:
    print("This is a small company.")
elif num_employees < 250:
    print("This is a medium-sized company.")
elif num_employees >= 250:
    print("This is a large company.")
```

Listing 3.4: โปรแกรมตรวจสอบขนาดบริษัทจากจำนวนพนักงาน

ตัวอย่างที่ 2:

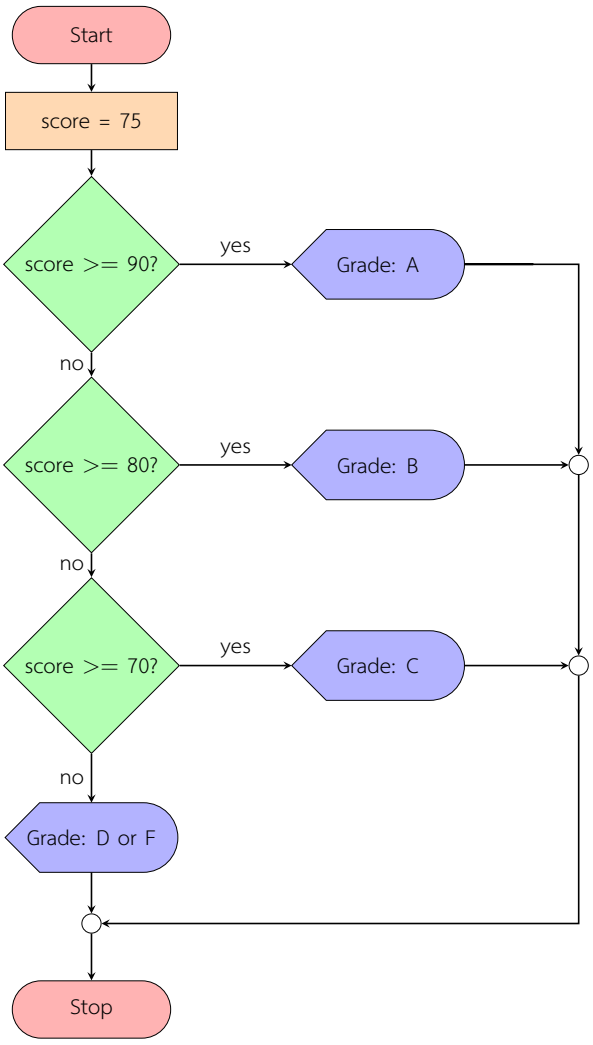

Figure 3.4: ผังงานสำหรับการประเมินผลเกรด

```
score = 75
if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: D or F")
```

Listing 3.5: Example of Elif condition

3.1.3 คำสั่ง Else

คำสั่ง `else` ช่วยให้คุณสามารถดำเนินการกลุ่มคำสั่งได้เมื่อไม่มีเงื่อนไขใดก่อนหน้านี้เป็นจริง โดยทำหน้าที่เป็นตัวเลือกสุดท้ายสำหรับสถานการณ์ที่ไม่ตรงกับเกณฑ์ที่กำหนดไว้ ตำแหน่งของ `else` อยู่ท้ายสุดของโครงสร้าง `if-elif` ซึ่งทำให้มั่นใจได้ว่ามีการกำหนดเส้นทางการทำงานของโปรแกรมอย่างชัดเจนในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จ สิ่งนี้รับประกันว่าโปรแกรมสามารถจัดการกับสถานการณ์ที่ไม่คาดคิดหรือกรณีเริ่มต้นได้อย่างราบรื่น โดยไม่ละเว้นเงื่อนไขใด ๆ เช่น ในการจัดประเภทคะแนนเป็นเกรดตัวอักษร `else` สามารถใช้ในการกำหนดเกรดตก หากไม่เข้าเกณฑ์ระดับที่สูงกว่าเลย บล็อกคำสั่งสุดท้ายนี้ช่วยให้สามารถจัดการกับกรณีพิเศษได้ และทำให้การไหลของโปรแกรมมีความเสถียรและน่าเชื่อถือมากยิ่งขึ้น คำสั่ง `else` ยังช่วยเสริมความยืดหยุ่นและความสมบูรณ์ของโครงสร้างเงื่อนไข โดยให้ทางเลือกสำรองที่ทำให้โปรแกรมสามารถตอบสนองต่อช่วงของข้อมูลนำเข้าได้หลากหลาย และรักษาพฤติกรรมที่สอดคล้องกันได้

```
if condition1:
    # code to execute if condition1 is true
elif condition2:
    # code to execute if condition2 is true
else:
    # code to execute if none of the above conditions are true
```

Listing 3.6: Syntax of Else condition

```
temperature = 30
if temperature > 30:
    print("It's hot outside.")
elif temperature > 20:
    print("The weather is nice.")
else:
    print("It's cold outside.")
```

Listing 3.7: Example of Else condition

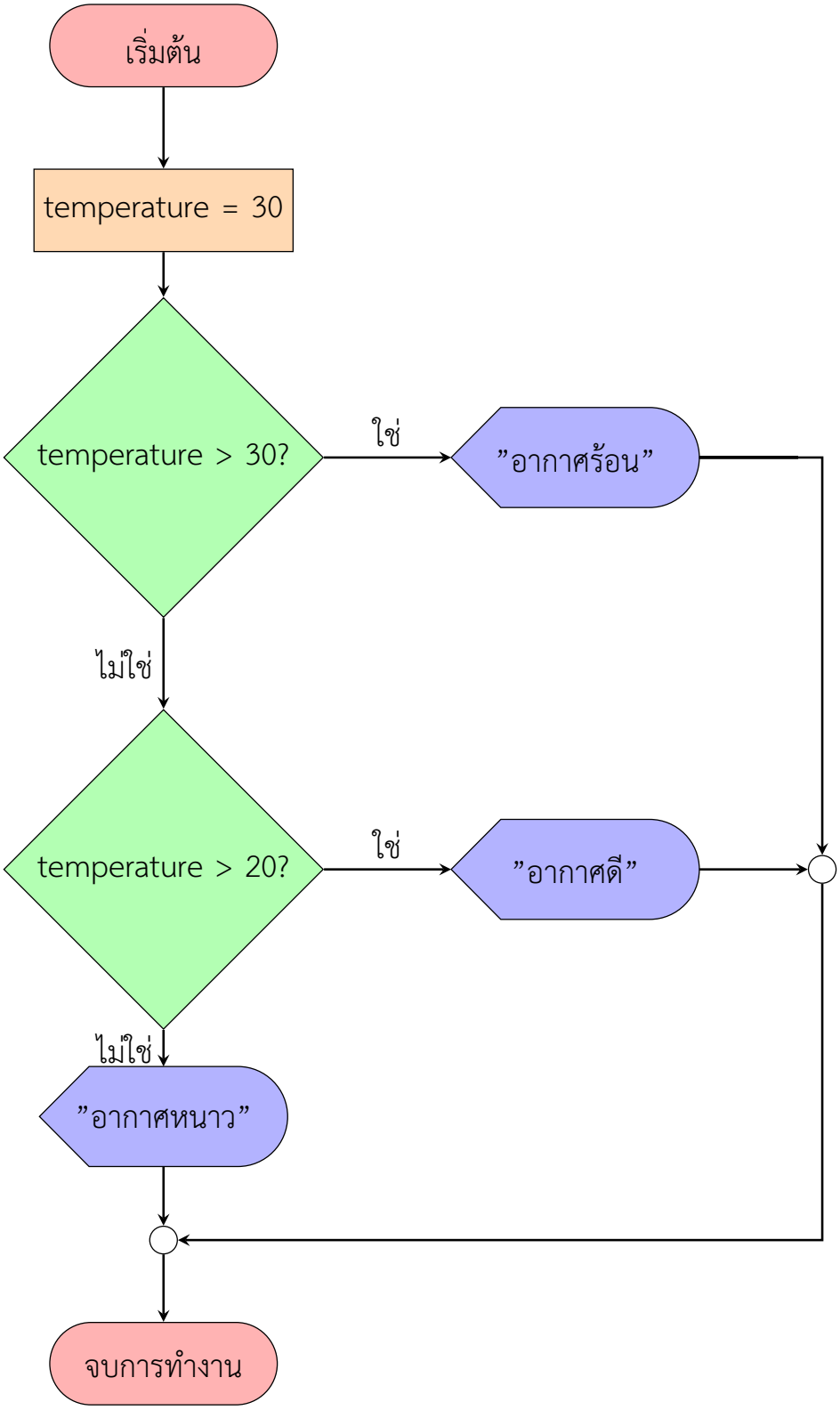

Figure 3.5: ผังงานสำหรับโปรแกรมตรวจสอบอุณหภูมิ

Algorithm 7: อัลกอริธึมสำหรับจำแนกประเภทตัวอักษร (แบบฝึกหัด)

Input: ตัวอักษรที่ผู้ใช้ป้อน: `inchar`
Output: ข้อความแสดงประเภทของตัวอักษร

```
1  begin
2      แสดง "กรุณาป้อนตัวอักษรหนึ่งตัว:";
3      inchar = input();
4      if inchar ≥ 'A' และ inchar ≤ 'Z' then
5          แสดง "คุณป้อนอักษรตัวพิมพ์ใหญ่", inchar;
6      else
7          if inchar ≥ 'a' และ inchar ≤ 'z' then
8              แสดง "คุณป้อนอักษรตัวพิมพ์เล็ก", inchar;
9          else
10             if inchar ≥ '0' และ inchar ≤ '9' then
11                 แสดง "คุณป้อนตัวเลข", inchar;
12             else
13                 แสดง "ไม่ใช่ตัวอักษรหรือตัวเลข", inchar;
```

3.1.4 Nested If

เงื่อนไขซ้อนในโปรแกรมหมายถึงการวางคำสั่ง `if` หรือ `elif` ไว้ภายในบล็อก `if`, `elif` หรือ `else` อื่น โครงสร้างลักษณะนี้ช่วยให้สามารถประมวลผลเงื่อนไขหลายระดับได้อย่างมีประสิทธิภาพ ถึงแม้ว่าเงื่อนไขซ้อนจะสามารถรองรับตรรกะที่ซับซ้อนได้ แต่ก็อาจทำให้โค้ดอ่านยากขึ้นและบำรุงรักษายากขึ้น เนื่องจากระดับการเยื้องบรรทัดที่เพิ่มขึ้น ดังนั้น การใช้เงื่อนไขซ้อนควรทำด้วยความระมัดระวัง เพื่อให้โค้ดยังคงความชัดเจนและสามารถดูแลได้ง่าย

```python
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Listing 3.8: Example of Number Categorization I

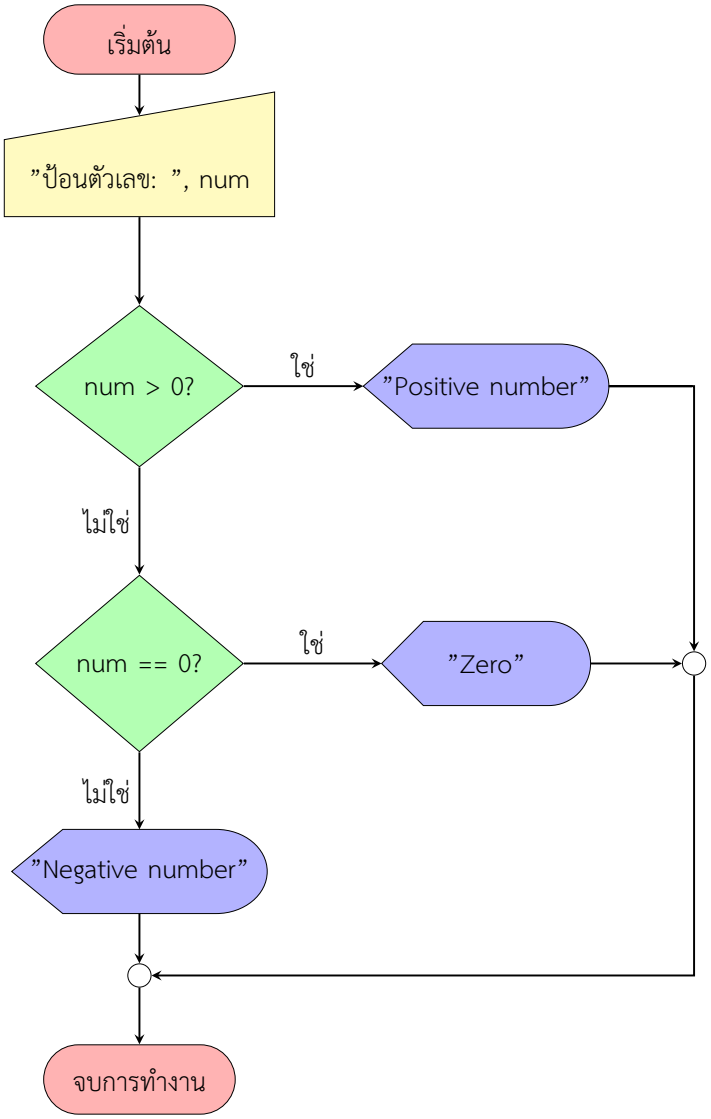

Figure 3.6: ผังงานสำหรับการจัดหมวดตัวเลข I

ตัวอย่าง:

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Listing 3.9: Example of Number Categorization II

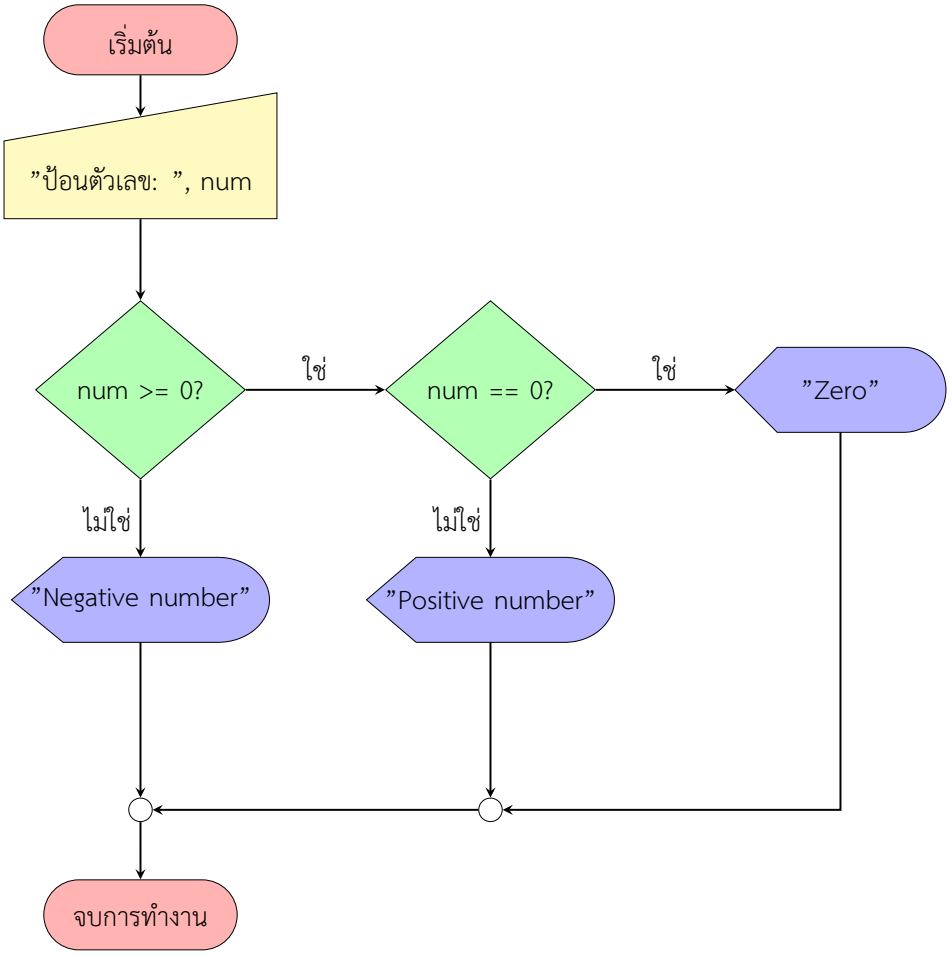

Figure 3.7: ผังงานสำหรับการจัดหมวดตัวเลข II

ข้อดีและข้อจำกัด

การจัดหมวดตัวเลขแบบที่ I: ข้อดี:

- **ความชัดเจน:** โค้ดนี้อ่านง่าย แยกแยะค่าบวก ศูนย์ และค่าลบได้ชัดเจนโดยใช้คำสั่ง `if`, `elif` และ `else` แยกกัน
- **เรียบง่าย:** โครงสร้างมีความเรียบง่ายและเข้าใจง่าย เหมาะสำหรับผู้เริ่มต้น

ข้อจำกัด:

- **ความซ้ำซ้อน:** เงื่อนไข `num == 0` ถูกตรวจสอบแยกจาก `num > 0` ซึ่งในบางกรณีอาจถือเป็นการซ้ำซ้อน
- **ผลกระทบด้านประสิทธิภาพ:** แม้จะเล็กน้อยสำหรับโปรแกรมขนาดเล็ก แต่การมีเงื่อนไขแยกอาจกระทบต่อประสิทธิภาพในระบบขนาดใหญ่

การจัดหมวดตัวเลขแบบที่ II: ข้อดี:

- **ประสิทธิภาพ:** การรวมเงื่อนไข `num >= 0` แล้วตรวจสอบ `num == 0` ภายใน ช่วยลดจำนวนการตรวจสอบแยก ทำให้ประสิทธิภาพดีขึ้นเล็กน้อย
- **การจัดกลุ่มตามตรรกะ:** การจัดกลุ่มการตรวจสอบค่าบวกและศูนย์เข้าด้วยกัน ช่วยให้การไหลของตรรกะดูเป็นระบบมากขึ้นสำหรับบางคน

ข้อจำกัด:

- **เงื่อนไขซ้อน:** การใช้คำสั่ง `if` ซ้อนกันทำให้โค้ดอ่านยาก โดยเฉพาะสำหรับผู้เริ่มต้น และทำให้เกิดการเยื้องที่ซับซ้อน
- **ความซับซ้อน:** โครงสร้างที่ซ้อนกันทำให้โค้ดดูซับซ้อนมากกว่าแบบแรก แม้ว่าจะทำงานเหมือนกัน

สรุป:

ทั้งสองแนวทางสามารถใช้งานได้และให้ผลลัพธ์เหมือนกัน การเลือกใช้อย่างใดอย่างหนึ่งขึ้นอยู่กับบริบทและความชอบของผู้พัฒนา โปรแกรมแบบแรกอ่านง่ายและตรงไปตรงมา เหมาะกับผู้เริ่มต้น ส่วนแบบที่สองมีความซับซ้อนเล็กน้อยแต่ให้ประสิทธิภาพที่สูงขึ้น เหมาะสำหรับผู้มีประสบการณ์ที่ต้องการโค้ดที่จัดระเบียบตามตรรกะมากขึ้น

3.2 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ (Comparison Operators) ในการเขียนโปรแกรมใช้เพื่อเปรียบเทียบค่าระหว่างสองค่าข้อมูล และส่งค่ากลับเป็น `True` หรือ `False` ขึ้นอยู่กับผลลัพธ์ของการเปรียบเทียบ ตัวดำเนินการที่ใช้บ่อย ได้แก่ `==` (เท่ากับ), `!=` (ไม่เท่ากับ), `>` (มากกว่า), `<` (น้อยกว่า), `>=` (มากกว่าหรือเท่ากับ) และ `<=` (น้อยกว่าหรือเท่ากับ) ตัวดำเนินการเหล่านี้มีความสำคัญในการควบคุมการไหลของโปรแกรม เช่น คำสั่ง `if age > 18:` จะตรวจสอบว่าอายุมากกว่า 18 หรือไม่ การเข้าใจตัวดำเนินการเปรียบเทียบจึงเป็นสิ่งจำเป็นในการพัฒนาโปรแกรมที่ตอบสนองต่อเงื่อนไขต่าง ๆ

แนวคิดสำคัญของตัวดำเนินการเปรียบเทียบ:

- **ตัวดำเนินการเปรียบเทียบ:** ใช้ภายในเงื่อนไขเพื่อเปรียบเทียบค่า เช่น `==`, `!=`, `>`, `<`, `>=`, `<=`
- **คำสั่งตามเงื่อนไข:** มักใช้งานร่วมกับ `if`, `elif`, และ `else` เพื่อควบคุมการทำงานของโปรแกรม
- **การเปรียบเทียบตัวเลขและข้อความ:** ใช้ได้ทั้งกับค่าตัวเลขและข้อความ
- **การตัดสินใจ:** เป็นเครื่องมือสำคัญในการตัดสินใจและสั่งให้โปรแกรมทำงานตามเงื่อนไขที่กำหนด

Table 3.1: การเปรียบเทียบตัวดำเนินการ

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
$>$	มากกว่า - คืนค่า `True` ถ้าค่าทางซ้ายมากกว่าค่าทางขวา	$x > y$
$<$	น้อยกว่า - คืนค่า `True` ถ้าค่าทางซ้ายน้อยกว่าค่าทางขวา	$x < y$
$==$	เท่ากับ - คืนค่า `True` ถ้าทั้งสองค่ามีค่าเท่ากัน	$x == y$
$!=$	ไม่เท่ากัน - คืนค่า `True` ถ้าค่าทั้งสองไม่เท่ากัน	$x! = y$
$>=$	มากกว่าหรือเท่ากับ - คืนค่า `True` ถ้าค่าทางซ้ายมากกว่าหรือเท่ากับขวา	$x >= y$
$<=$	น้อยกว่าหรือเท่ากับ - คืนค่า `True` ถ้าค่าทางซ้ายน้อยกว่าหรือเท่ากับขวา	$x <= y$

3.2.1 การเปรียบเทียบค่า

การเปรียบเทียบค่าในภาษา Python เกิดจากการใช้ตัวดำเนินการเพื่อประเมินความสัมพันธ์ระหว่างสองค่า ผลลัพธ์ที่ได้จะเป็นค่า Boolean (`True` หรือ `False`) ซึ่งมีความสำคัญในการตัดสินใจให้โปรแกรมทำงานตามเงื่อนไขที่กำหนดไว้

ตัวอย่าง:

```
x = 10
y = 20

print(x == y)  # False
print(x != y)  # True
print(x > y)   # False
print(x < y)   # True
print(x >= y)  # False
print(x <= y)  # True
```

Listing 3.10: Example of Comparison Operators

3.2.2 การเปรียบเทียบข้อความ

การเปรียบเทียบข้อความใน Python ทำโดยการตรวจสอบลำดับทางพจนานุกรม (lexicographical order) หรือการเปรียบเทียบความเท่ากันของข้อความ โดยใช้ตัวดำเนินการเช่น `==`, `!=`, `<`, `>`, `<=`, และ `>=` ซึ่งเปรียบเทียบตามค่า Unicode ของตัวอักษรแต่ละตัว

ตัวอย่าง เช่น `if "apple" == "apple":` ตรวจสอบว่าข้อความทั้งสองเหมือนกันหรือไม่ หรือ `if "apple" < "banana":` เปรียบเทียบตามลำดับอักษร 'a' กับ 'b' ซึ่ง 'a' มีค่าน้อยกว่า 'b' ดังนั้นผลลัพธ์จะเป็น `True`

นอกจากนี้ Python ยังรองรับการเปรียบเทียบแบบไม่สนใจตัวพิมพ์ใหญ่หรือพิมพ์เล็ก โดยใช้เมธอด `lower()` หรือ `upper()` เช่น `"Apple".lower() == "apple".lower()` เพื่อให้ผลลัพธ์สอดคล้องกันโดยไม่คำนึงถึงตัวพิมพ์

ตัวอย่าง:

```
#             "Mary"    "Mark"

#
string1 = "Mary"
string2 = "Mark"

#
if string1 == string2:
    print(f'"{string1}" and "{string2}" are equal.')
else:
    print(f'"{string1}" and "{string2}" are not equal.')

#
if string1 < string2:
    print(f'"{string1}" comes before "{string2}" in lexicographical order.'
        )
elif string1 > string2:
    print(f'"{string1}" comes after "{string2}" in lexicographical order.')

#               /
if string1.lower() == string2.lower():
    print(f'"{string1}" and "{string2}" are equal when case is ignored.')
else:
    print(f'"{string1}" and "{string2}" are not equal when case is ignored.
        ')
```

Listing 3.11: โปรแกรมเปรียบเทียบข้อความ ”Mary” และ ”Mark”

ในโปรแกรมนี้:

- กำหนดข้อความสองตัวคือ `string1` และ `string2` มีค่าเป็น `"Mary"` และ `"Mark"`
- เปรียบเทียบข้อความว่าเท่ากันหรือไม่โดยใช้ตัวดำเนินการ `==`
- เปรียบเทียบลำดับของข้อความโดยใช้ตัวดำเนินการ `<` และ `>`
- เปรียบเทียบแบบไม่สนใจตัวพิมพ์โดยใช้ `lower()` เพื่อความถูกต้องสม่ำเสมอ

3.3 ตัวดำเนินการทางตรรกะ

ตัวดำเนินการทางตรรกะในภาษาโปรแกรมใช้สำหรับรวมหลายเงื่อนไขเข้าด้วยกันเพื่อสร้างตรรกะการตัดสินใจที่ซับซ้อน ตัวดำเนินการหลัก ได้แก่ `and`, `or`, และ `not` โดย `and` จะให้ค่าเป็น True ก็ต่อเมื่อทุกเงื่อนไขเป็นจริง, `or` จะให้ค่าเป็น True หากมีอย่างน้อยหนึ่งเงื่อนไขเป็นจริง และ `not` จะกลับค่าความจริงของเงื่อนไข กล่าวคือ True จะกลายเป็น False และในทางกลับกัน ตัวดำเนินการเหล่านี้มีความสำคัญอย่างยิ่งในการเขียนโปรแกรมที่ต้องพิจารณาหลายเงื่อนไขและตัดสินใจอย่างละเอียดอ่อนตามข้อมูลที่ได้รับ

แนวคิดสำคัญของตัวดำเนินการทางตรรกะ:

- **Logical Operators:** ใช้สำหรับรวมเงื่อนไขหลายเงื่อนไขโดยใช้ `and`, `or`, `not`
- **Combining Conditions:** ใช้สร้างนิพจน์ทางตรรกะที่ซับซ้อนโดยรวมเงื่อนไขหลายประการ
- **Boolean Logic:** ทำงานกับค่าความจริง (`True` หรือ `False`)
- **Conditional Control:** เพิ่มศักยภาพการตัดสินใจภายในคำสั่ง `if`, `elif`, และ `else`
- **Short-Circuit Evaluation:** หยุดการประเมินเงื่อนไขเมื่อทราบผลลัพธ์แล้ว (เช่น `and` จะไม่ประเมินเงื่อนไขที่สองหากเงื่อนไขแรกเป็น False)
- **Clarity and Efficiency:** ช่วยให้เขียนโค้ดได้ชัดเจนและมีประสิทธิภาพโดยการรวมการตรวจสอบหลายรายการให้อยู่ในคำสั่งเดียว

And (`and`): คืนค่า True ถ้าเงื่อนไขทั้งสองเป็นจริง

```
a and b
```

Or (`or`): คืนค่า True ถ้ามีเงื่อนไขอย่างน้อยหนึ่งเป็นจริง

```
a or b
```

Not (`not`): คืนค่า True ถ้าเงื่อนไขเป็นเท็จ

```
not a
```

Table 3.2: เปรียบเทียบตัวดำเนินการทางตรรกะ

Operator	ความหมาย
and	**And (`and`):** คืนค่า True ถ้าเงื่อนไขทั้งสองเป็นจริง
or	**Or (`or`):** คืนค่า True ถ้ามีเงื่อนไขอย่างน้อยหนึ่งเป็นจริง
not	**Not (`not`):** คืนค่า True ถ้าเงื่อนไขเป็นเท็จ

ตัวอย่าง:

```
x = 10
y = 20
z = 30

# Using 'and' operator
if x < y and y < z:
    print("x is less than y and y is less than z.")  # True

# Using 'or' operator
if x < y or y > z:
    print("Either x is less than y or y is greater than z.")  # True

# Using the 'not' operator
if not (x > y):
    print("x is not greater than y.")  # True
```

Listing 3.12: Example of Logical Operators

3.4 ตัวดำเนินการเปรียบเทียบอัตลักษณ์ (Identity Operators)

ตัวดำเนินการเปรียบเทียบอัตลักษณ์ในภาษา Python คือ `is` และ `is not` ใช้เพื่อตรวจสอบว่าสองตัวแปรชี้ไปยังออบเจกต์เดียวกันในหน่วยความจำหรือไม่ ต่างจากตัวดำเนินการเปรียบเทียบค่า (==) ที่ใช้ตรวจสอบค่าของตัวแปร ตัวดำเนินการเปรียบเทียบอัตลักษณ์ใช้เพื่อตรวจสอบว่าอ้างอิงถึงออบเจกต์เดียวกันหรือไม่ เช่น `a is b` จะคืนค่า `True` หาก `a` และ `b` ชี้ไปยังออบเจกต์เดียวกัน ในขณะที่ `a is not b` จะคืนค่า `True` หากชี้ไปยังออบเจกต์คนละตัว ตัวดำเนินการนี้เหมาะสำหรับการเปรียบเทียบออบเจกต์เพื่อให้แน่ใจว่าทั้งสองอ้างอิงตำแหน่งในหน่วยความจำเดียวกัน ซึ่งช่วยให้มั่นใจถึงความถูกต้องในการเปรียบเทียบข้อมูลเชิงออบเจกต์

Table 3.3: เปรียบเทียบตัวดำเนินการเปรียบเทียบอัตลักษณ์

Operator	ความหมาย	ตัวอย่าง
is	คืนค่า True หากตัวแปรทั้งสองชี้ไปยังออบเจกต์เดียวกัน	`x is True`
is not	คืนค่า True หากตัวแปรทั้งสองไม่ชี้ไปยังออบเจกต์เดียวกัน	`x is not True`

```python
# Example of the identity operator

# Two variables pointing to the same list object
a = [1, 2, 3]
b = a

# Two variables pointing to different list objects with the same content
c = [1, 2, 3]
d = [1, 2, 3]

# Using the identity operator
print(a is b)  # True, since a and b refer to the same object
print(a is c)  # False, since a and c refer to different objects
print(c is d)  # False, since c and d refer to different objects

# Using the equality operator for comparison
print(a == c)  # True, since the contents of a and c are equal
print(c == d)  # True, since the contents of c and d are equal
```

Listing 3.13: Example of the identity operator

3.5 ตัวดำเนินการตรวจสอบสมาชิก(Membership Operators)

ตัวดำเนินการตรวจสอบสมาชิกในภาษา Python ได้แก่ `in` และ `not in` ใช้เพื่อตรวจสอบว่าค่าหรือตัวแปรนั้นอยู่ภายในลำดับข้อมูล เช่น string, list, tuple หรือ set หรือไม่ ตัวดำเนินการ `in` จะคืนค่า `True` หากค่าที่ระบุพบอยู่ในลำดับ ในขณะที่ `not in` จะคืนค่า `True` หากค่านั้นไม่อยู่ในลำดับ ตัวดำเนินการเหล่านี้มีประโยชน์มากในการตรวจสอบการมีอยู่ของสมาชิก และช่วยให้สามารถเขียนเงื่อนไขต่าง ๆ ได้อย่างกระชับและชัดเจนเมื่อทำงานกับโครงสร้างข้อมูล เช่น `x in list` หมายถึงการตรวจสอบว่า `x` เป็นสมาชิกของ `list` หรือไม่ ความสามารถนี้ช่วยเพิ่มประสิทธิภาพและความชัดเจนของโค้ดเมื่อจัดการกับข้อมูลจำนวนมาก

Table 3.4: ตัวดำเนินการตรวจสอบสมาชิก

Operator	ความหมาย	ตัวอย่าง
in	คืนค่า True หากค่าหรือตัวแปรพบในลำดับ	`5 in x`
not in	คืนค่า True หากค่าหรือตัวแปรไม่พบในลำดับ	`5 not in x`

```python
# Example of membership operators

# List of fruits
fruits = ["apple", "banana", "cherry"]

# Using 'in' operator
print("banana" in fruits)  # True, since "banana" is in the list
print("orange" in fruits)  # False, since "orange" is not in the list

# Using 'not in' operator
print("grape" not in fruits)  # True, since "grape" is not in the list
print("apple" not in fruits)  # False, since "apple" is in the list

# String example
sentence = "The quick brown fox jumps over the lazy dog."
print("fox" in sentence)  # True, since "fox" is a substring of the
    sentence
print("cat" not in sentence)  # True, since "cat" is not a substring of the
     sentence
```

Listing 3.14: Example of membership operators

3.6 การรวมตัวดำเนินการเปรียบเทียบและตรรกะ(Combining Comparison and Logical Operators)

เราสามารถรวมตัวดำเนินการเปรียบเทียบเข้ากับตัวดำเนินการทางตรรกะเพื่อสร้างเงื่อนไขที่ซับซ้อน ซึ่งทำให้โปรแกรมสามารถตัดสินใจได้อย่างแม่นยำและยืดหยุ่นมากขึ้น โดยการใช้ตัวดำเนินการเปรียบเทียบ เช่น ==, !=, >, <, >=, และ <= เพื่อประเมินความสัมพันธ์ระหว่างค่าต่าง ๆ และเมื่อนำมาใช้ร่วมกับตัวดำเนินการทางตรรกะ เช่น `and`, `or`, และ `not` ก็จะสามารถตรวจสอบหลายเงื่อนไขพร้อมกันได้ และดำเนินการตามเงื่อนไขที่ซับซ้อน

ตัวอย่างเช่น อาจต้องการตรวจสอบว่าค่าหนึ่งอยู่ในช่วงที่กำหนด และมีสถานะสมาชิกที่ถูกต้องด้วย การรวมเงื่อนไขในลักษณะนี้ช่วยให้สามารถควบคุมการทำงานของโปรแกรมได้อย่างแม่นยำและยืดหยุ่น เหมาะสำหรับโปรแกรมที่ต้องรองรับการตัดสินใจที่มีความซับซ้อน การเข้าใจและใช้ตัวดำเนินการเหล่านี้อย่างมีประสิทธิภาพเป็นพื้นฐานสำคัญในการพัฒนาโปรแกรมที่เชื่อถือได้และตอบสนองต่อสถานการณ์ได้ดี

```python
age = 25
income = 50000

# Check if the age is between 18 and 65 and income is above 30000
if age >= 18 and age <= 65 and income > 30000:
    print("You are eligible for the loan.")
else:
    print("You are not eligible for the loan.")
```

Listing 3.15: Example of Logical Operators

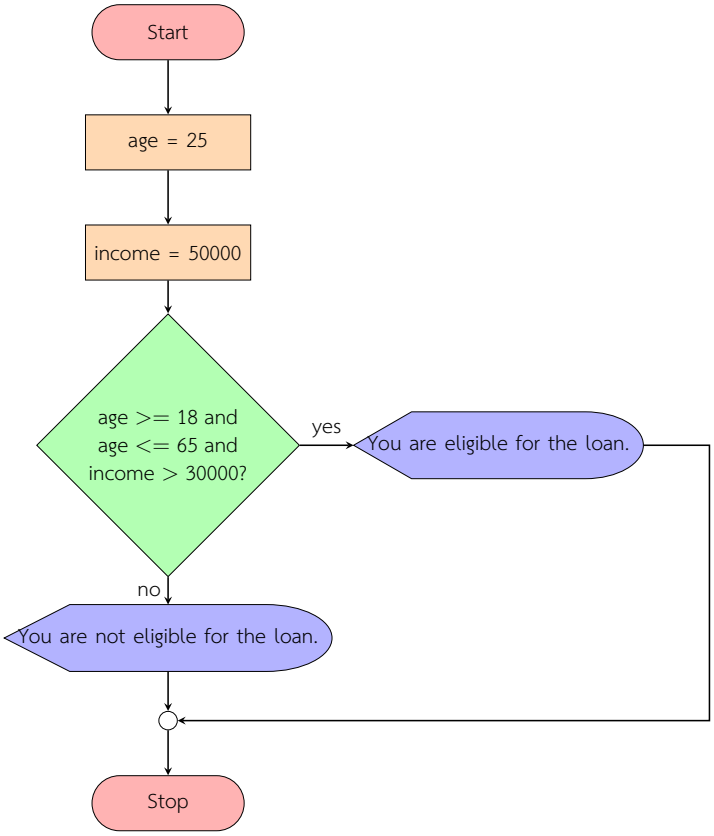

Figure 3.8: ผังงานสำหรับตรวจสอบสิทธิ์การขอสินเชื่อ

3.7 ตัวอย่างการใช้งานจริง

3.7.1 โปรแกรมตรวจสอบสถานะการเข้าศึกษา

โปรแกรมนี้แสดงการประยุกต์ใช้เงื่อนไขที่ขึ้นกับอายุและคะแนนสอบเพื่อประเมินสถานะการเข้าศึกษา โดยรับค่าจากผู้ใช้เป็นอายุและคะแนนสอบ แล้วใช้โครงสร้างเงื่อนไขเพื่อตรวจสอบว่าทั้งสองเงื่อนไขตรงกับเกณฑ์หรือไม่ ผู้สมัครจะได้รับการตอบรับหากมีอายุระหว่าง 18 ถึง 25 ปี และมีคะแนนสอบมากกว่าหรือเท่ากับ 70 หากไม่ตรงตามเงื่อนไขใดเงื่อนไขหนึ่ง จะไม่ได้รับการตอบรับ วิธีนี้ช่วยให้พิจารณาข้อมูลอย่างรอบด้านในการตัดสินใจ

```
# Program to determine the admission status
# based on age and test score
age = int(input("Enter your age: "))
test_score = int(input("Enter your test score: "))

if age >= 18 and age <= 25 and test_score >= 70:
    print("Congratulations! You are admitted to the program.")
else:
    print("Sorry, you do not meet the admission criteria.")
```

Listing 3.16: Example of Logical Operators

3.7.2 โปรแกรมตรวจสอบผ่านหรือไม่ผ่าน

อัลกอริทึมนี้เริ่มจากการให้ผู้ใช้ป้อนคะแนน แล้วแปลงเป็นตัวเลขจำนวนเต็ม จากนั้นใช้คำสั่งเงื่อนไขเพื่อตรวจสอบว่าคะแนนมากกว่าหรือเท่ากับ 50 หรือไม่ หากเป็นจริงจะแสดงผลว่า “ผ่าน” หากไม่เป็นจริงจะแสดงผลว่า “ไม่ผ่าน” วิธีนี้ช่วยให้สามารถประเมินผลผ่าน/ไม่ผ่านได้อย่างแม่นยำตามเกณฑ์ที่กำหนดไว้

Algorithm 8: อัลกอริทึมเพื่อตรวจสอบผ่านหรือไม่ผ่าน

Input: score
Output: ข้อความผ่านหรือไม่ผ่าน

```
1 begin
2     Read score ;
3     if score >= 50 then
4         Display 'Pass';
5     else
6         Display 'Fail';
```

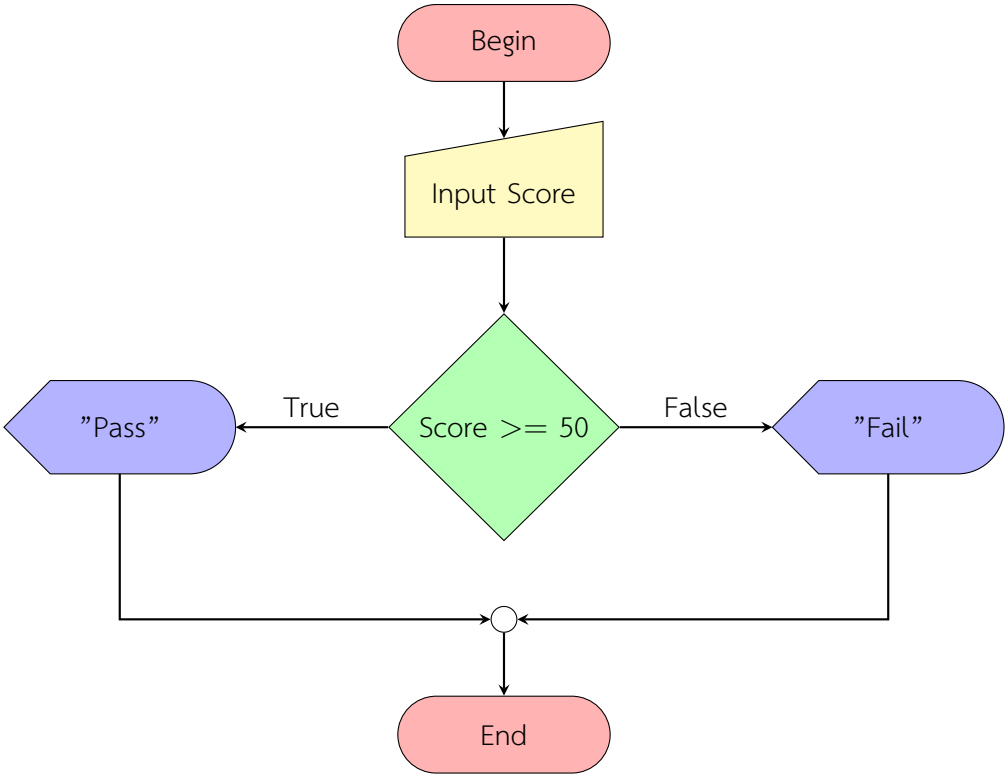

Figure 3.9: ผังงานสำหรับการประเมินผลคะแนน

บทที่ 3 โจทย์และแบบฝึกหัด: คำสั่งเงื่อนไขและการควบคุมการไหลของโปรแกรม

3.1 ตรวจสอบว่าตัวเลขเป็นบวกหรือไม่
เขียนโปรแกรมเพื่อตรวจสอบว่าตัวเลขเป็นบวกหรือไม่ หากเป็นบวก ให้พิมพ์ว่า "The number is positive."

3.2 ตรวจสอบคุณสมบัติผู้มีสิทธิเลือกตั้ง
เขียนโปรแกรมเพื่อตรวจสอบว่าผู้ใช้มีสิทธิเลือกตั้งหรือไม่ โดยผู้มีสิทธิเลือกตั้งต้องมีอายุ 18 ปีขึ้นไป

3.3 ตรวจสอบว่าตัวเลขเป็นเลขคู่หรือคี่
เขียนโปรแกรมเพื่อตรวจสอบว่าตัวเลขเป็นเลขคู่หรือคี่ ให้พิมพ์ "Even" หากเป็นเลขคู่ และ "Odd" หากเป็นเลขคี่

3.4 กำหนดเกรดตามคะแนนที่ได้
เขียนโปรแกรมเพื่อกำหนดเกรดจากคะแนนที่ได้รับ โดยใช้เกณฑ์ดังนี้: A สำหรับคะแนน $\geq$ 90, B สำหรับคะแนน $\geq$ 80, C สำหรับคะแนน $\geq$ 70 และ D สำหรับคะแนน $<$ 70

3.5 ตรวจสอบอุณหภูมิและแสดงข้อความ
เขียนโปรแกรมเพื่อตรวจสอบอุณหภูมิและแสดงข้อความ หากอุณหภูมิมากกว่า 30 ให้พิมพ์ "It's hot outside." หากอยู่ระหว่าง 20 ถึง 30 ให้พิมพ์ "The weather is nice." หากต่ำกว่า 20 ให้พิมพ์ "It's cold outside."

3.6 ตรวจสอบว่าตัวเลขสองตัวเท่ากันหรือไม่
เขียนโปรแกรมเพื่อตรวจสอบว่าตัวเลขสองตัวเท่ากันหรือไม่ ให้พิมพ์ "Equal" หากเท่ากัน มิฉะนั้นให้พิมพ์ "Not Equal."

3.7 ตรวจสอบสิทธิส่วนลดสำหรับผู้สูงอายุ
เขียนโปรแกรมเพื่อตรวจสอบว่าบุคคลมีสิทธิได้รับส่วนลดผู้สูงอายุหรือไม่ โดยมีสิทธิหากมีอายุ 65 ปีขึ้นไป

3.8 ตรวจสอบว่าตัวเลขอยู่ในช่วงที่กำหนดหรือไม่
เขียนโปรแกรมเพื่อตรวจสอบว่าตัวเลขอยู่ในช่วงที่กำหนดหรือไม่ ให้พิมพ์ "Within range" หากตัวเลขอยู่ระหว่าง 10 ถึง 20 (รวมทั้งสองขอบเขต) มิฉะนั้นให้พิมพ์ "Out of range."

3.9 ตรวจสอบว่าบุคคลเป็นวัยรุ่นหรือไม่
เขียนโปรแกรมเพื่อตรวจสอบว่าบุคคลเป็นวัยรุ่นหรือไม่ โดยถือว่าเป็นวัยรุ่นหากมีอายุระหว่าง 13 ถึง 19 ปี (รวมทั้งสองขอบเขต)

3.10 กำหนดสถานะการเข้าศึกษาตามอายุและคะแนนสอบ
เขียนโปรแกรมเพื่อตัดสินสถานะการเข้าศึกษาตามอายุและคะแนนสอบ ให้ตอบรับหากบุคคลมีอายุระหว่าง 18 ถึง 25 ปี และมีคะแนนสอบ 70 ขึ้นไป

ภาคผนวก

ภาคผนวกนี้ประกอบด้วยแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างโปรแกรม เพื่อเสริมความเข้าใจเกี่ยวกับคำสั่งเงื่อนไขและการควบคุมการไหลของโปรแกรมที่ได้เรียนรู้ในบทที่ 3

A3.1 แหล่งข้อมูลเพิ่มเติม (Additional Resources)

แนะนำหนังสือและคอร์สออนไลน์สำหรับศึกษาคำสั่งเงื่อนไขเพิ่มเติม

- หนังสือ:
  - *Python Crash Course* โดย Eric Matthes
  - *Effective Python: 90 Specific Ways to Write Better Python* โดย Brett Slatkin
- บทเรียนออนไลน์:
  - Python Conditions - W3Schools
  - Control Flow in Python - Real Python
- คอร์สออนไลน์:
  - Coursera: Python for Data Science and AI โดย IBM
  - Udemy: Complete Python Bootcamp: Go from zero to hero in Python 3

A3.2 แหล่งอ้างอิง (References)

แหล่งอ้างอิงที่ใช้ในการจัดทำเนื้อหาและสามารถศึกษาเพิ่มเติมได้

- Python Software Foundation. (2024). *Python Documentation - Control Flow*. https://docs.python.org/3/tutorial/controlflow.html
- Sweigart, A. (2015). *Automate the Boring Stuff with Python*. No Starch Press.

A3.3 แบบฝึกหัดเสริม (Exercises)

ฝึกฝนการใช้คำสั่งเงื่อนไขพื้นฐานและขั้นสูง พร้อมการประยุกต์ใช้ตรรกะในโปรแกรม

- **Exercise 1: Basic Conditional Statements**
  - ตรวจสอบว่าจำนวนที่ผู้ใช้ป้อนเป็นบวก ลบ หรือศูนย์
  - พิจารณาจากอายุของผู้ใช้ แล้วพิมพ์ว่าเป็นเด็ก วัยรุ่น ผู้ใหญ่ หรือผู้สูงอายุ
- **Exercise 2: Advanced Conditionals**
  - ตรวจสอบว่าปีที่ผู้ใช้ป้อนเป็นปีอธิกสุรทินหรือไม่
  - แสดงเกรดที่สอดคล้องกับคะแนนที่ผู้ใช้ป้อน
- **Exercise 3: Combining Comparison and Logical Operators**
  - รับเลข 3 จำนวนจากผู้ใช้ แล้วพิมพ์เลขที่มากที่สุด
  - ตรวจสอบว่าสตริงที่ผู้ใช้ป้อนมีทั้งอักษรพิมพ์ใหญ่และพิมพ์เล็กหรือไม่

A3.4 ตัวอย่างโปรแกรมใช้งานจริง (Practical Examples)

ตัวอย่างการเขียนโปรแกรมด้วยคำสั่งเงื่อนไขแบบต่าง ๆ

- **Example 1: การใช้ If-Else Statement**

  โปรแกรมตรวจสอบว่าจำนวนที่ป้อนเป็นเลขคู่หรือเลขคี่

```python
number = int(input("Enter a number: "))
if number % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

Listing 3.17: Python program to check if a number is even or odd

- **Example 2: การใช้ Nested If Statement**

  โปรแกรมตรวจสอบฤดูกาลจากเลขเดือนที่ผู้ใช้ป้อน

```python
month = int(input("Enter the month number (1-12): "))
if month in [12, 1, 2]:
    print("It's winter.")
elif month in [3, 4, 5]:
    print("It's spring.")
elif month in [6, 7, 8]:
    print("It's summer.")
elif month in [9, 10, 11]:
    print("It's autumn.")
else:
    print("Invalid month number.")
```

Listing 3.18: Python script to determine the season based on the month number

- **Example 3: การใช้ Logical Operators**

  โปรแกรมตรวจสอบว่าปีที่ป้อนเป็นปีอธิกสุรทินหรือไม่

```python
year = int(input("Enter a year: "))
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Listing 3.19: Python script to check if a year is a leap year

บทที่ 4

การใช้ลูป For และ While

บทนี้จะกล่าวถึงภาพรวมของการใช้ลูปและการวนซ้ำในภาษา Python โดยครอบคลุมทั้งลูป `for` และ `while` พร้อมอธิบายการทำงานและไวยากรณ์ที่เกี่ยวข้อง ลูปเป็นเครื่องมือสำคัญที่ใช้ในการวนซ้ำกับลำดับข้อมูล เช่น list, tuple, dictionary, set หรือ string ซึ่งทำให้สามารถดำเนินการซ้ำ ๆ ได้อย่างมีประสิทธิภาพ ในทางกลับกัน ลูป `while` เหมาะสำหรับสถานการณ์ที่ไม่สามารถทราบจำนวนครั้งที่ต้องวนซ้ำล่วงหน้าได้ โดยจะทำซ้ำจนกว่ามีเงื่อนไขที่กำหนดเป็นเท็จ

เนื้อหาในบทนี้ยังได้กล่าวถึงคำสั่งควบคุมลูปที่สำคัญ เช่น `break`, `continue` และ `pass` คำสั่ง `break` ใช้สำหรับออกจากลูปทันทีเมื่อเงื่อนไขที่กำหนดเป็นจริง ซึ่งมีประโยชน์ในการเพิ่มประสิทธิภาพและหลีกเลี่ยงการวนซ้ำที่ไม่จำเป็น คำสั่ง `continue` ใช้สำหรับข้ามรอบปัจจุบันและไปยังรอบถัดไปของลูป เหมาะสำหรับกรณีที่ต้องการข้ามเงื่อนไขบางอย่าง ส่วนคำสั่ง `pass` ใช้เป็นตัวแทนของโค้ดที่ยังไม่เสร็จสมบูรณ์ โดยช่วยให้โค้ดไม่เกิดข้อผิดพลาดแม้จะยังไม่มีคำสั่งให้ทำงานในจุดนั้น

นอกจากนี้ บทเรียนยังมีตัวอย่างจริงเพื่อเสริมความเข้าใจ เช่น การวนซ้ำผ่าน list และ string การใช้ฟังก์ชัน `range` และการคำนวณผลรวมและแฟกทอเรียล ตัวอย่างเหล่านี้ไม่เพียงแสดงแนวคิด แต่ยังแสดงให้เห็นถึงประโยชน์ในทางปฏิบัติ ทำให้นักเรียนเข้าใจและสามารถนำไปประยุกต์ใช้ได้อย่างมีประสิทธิภาพ ซึ่งจะช่วยเสริมสร้างพื้นฐานที่มั่นคงในเรื่องลูปและการวนซ้ำ ซึ่งเป็นแนวคิดหลักในภาษา Python

4.1 บทนำสู่ลูป (s)

ลูปช่วยให้คุณสามารถสั่งให้โค้ดทำงานซ้ำ ๆ ได้เมื่อเงื่อนไขที่กำหนดเป็นจริง ซึ่งเป็นสิ่งสำคัญในการจัดการกับงานที่ต้องทำซ้ำอย่างมีประสิทธิภาพในโปรแกรม หากไม่มีลูป คุณจะต้องเขียนโค้ดเดิมซ้ำหลายครั้ง ซึ่งทำให้โค้ดยาวขึ้น อ่านยาก และดูแลรักษายาก ลูปจึงช่วยลดข้อผิดพลาดและประหยัดเวลาในการเขียนโปรแกรม

ลูปใน Python แบ่งออกเป็น 2 ประเภทหลัก คือ ลูป `for` และลูป `while` ลูป `for` เหมาะสำหรับกรณีที่ทราบจำนวนรอบของการทำงานล่วงหน้า โดยจะวนซ้ำตามลำดับของข้อมูล เช่น list, tuple, dictionary, set หรือ string เหมาะสำหรับการประมวลผลรายการข้อมูล เช่น การแสดงค่าทุกตัวใน list

ในขณะที่ลูป `while` เหมาะสำหรับกรณีที่จำนวนรอบของการทำงานไม่แน่นอน โดยจะทำงานซ้ำตราบใดที่เงื่อนไขที่กำหนดยังคงเป็นจริง ซึ่งเหมาะสำหรับสถานการณ์ที่ขึ้นอยู่กับการคำนวณหรือการป้อนข้อมูลจากผู้ใช้

การใช้ลูปช่วยเพิ่มประสิทธิภาพ อ่านง่าย และบำรุงรักษาได้ง่ายขึ้น โดยการห่อหุ้มงานที่ทำซ้ำไว้ภายในลูป ทำให้โค้ดมีโครงสร้างชัดเจน เข้าใจง่าย และสามารถแก้ไขหรือขยายได้ง่าย ลูปจึงเป็นเครื่องมือสำคัญที่โปรแกรมเมอร์ทุกคนควรใช้ในการเขียนโปรแกรมที่มีความยืดหยุ่นและเชื่อถือได้

แนวคิดสำคัญเกี่ยวกับลูป:

- **คำจำกัดความและวัตถุประสงค์**
  - ลูปทำให้โค้ดทำงานซ้ำได้ตามเงื่อนไขที่กำหนด
  - จำเป็นต่อการทำงานที่ซ้ำ ๆ และช่วยเพิ่มประสิทธิภาพของโค้ด
- **กรณีการใช้งานทั่วไป**
  - วนซ้ำใน list, string และ iterable อื่น ๆ
  - ใช้ฟังก์ชัน `range()` เพื่อสร้างลำดับตัวเลข
  - คำนวณผลรวม แฟกทอเรียล และการวนลูปใน dictionary
- **ลูปซ้อน (Nested Loops)**
  - ใช้ลูปหนึ่งซ้อนอยู่ในอีกลูปเพื่อจัดการการทำงานที่ซับซ้อน
- **ลูปไม่สิ้นสุด (Infinite Loops)**
  - ลูปที่ไม่มีจุดจบเนื่องจากเงื่อนไขไม่เคยเป็นเท็จ
  - ควรมั่นใจว่าเงื่อนไขจะกลายเป็นเท็จในที่สุดเพื่อหลีกเลี่ยงลูปไม่รู้จบ
- **ตัวอย่างการใช้งานจริง**
  - วนซ้ำผ่านชุดข้อมูล
  - คำนวณผลรวมและแฟกทอเรียล
  - วนผ่าน key และ value ใน dictionary
- **ประสิทธิภาพและความเข้าใจง่าย**
  - ลูปช่วยลดการเขียนโค้ดซ้ำ และทำให้โค้ดอ่านง่ายขึ้น
  - ทำให้โค้ดดูแลรักษาและดีบักได้ง่ายขึ้น

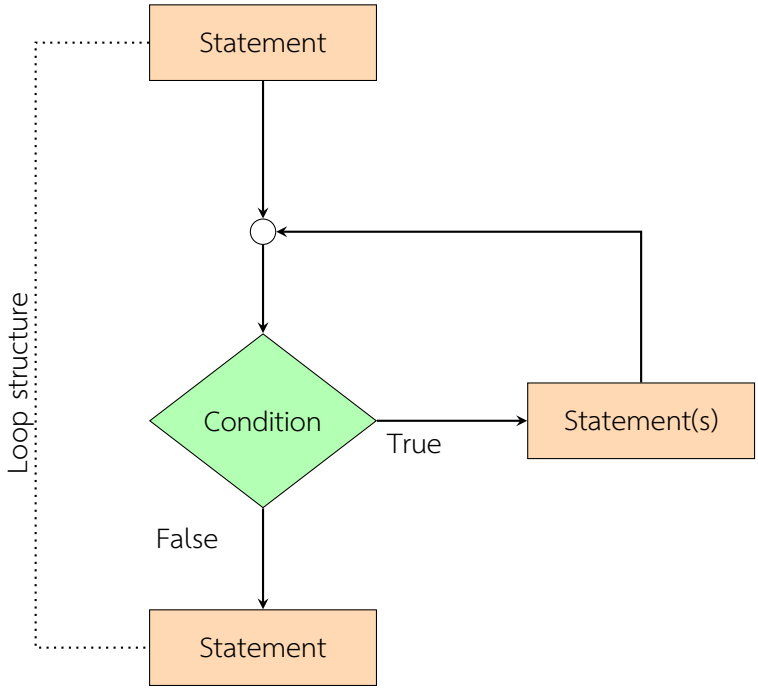

Figure 4.1: ผังงานของโครงสร้างลูป

4.2 ลูป For

ลูป `for` จะทำการวนซ้ำผ่านลำดับของข้อมูล ซึ่งรวมถึงโครงสร้างข้อมูลต่าง ๆ เช่น รายการ (list), ทูเพิล (tuple), ดิกชันนารี (dictionary), เซต (set), สตริง (string) และอ็อบเจกต์ที่สามารถวนซ้ำได้อื่น ๆ ซึ่งหมายความว่า ลูปจะวนผ่านแต่ละองค์ประกอบในลำดับโดยอัตโนมัติทีละรายการ ช่วยให้คุณสามารถดำเนินการต่าง ๆ กับแต่ละองค์ประกอบได้โดยไม่ต้องจัดการกับดัชนีเอง ตัวอย่างเช่น ในรายการของข้อมูล ลูป `for` สามารถเข้าถึงแต่ละรายการโดยตรงและดำเนินการโค้ดกับแต่ละรายการได้ ในทำนองเดียวกัน ลูป `for` สามารถวนซ้ำผ่านคีย์ ค่าหรือคู่คีย์-ค่าในดิกชันนารี ทำให้สามารถจัดการและดึงข้อมูลได้อย่างมีประสิทธิภาพ ความยืดหยุ่นของลูปประเภทนี้จึงมีความสำคัญอย่างยิ่งต่อภารกิจต่าง ๆ ตั้งแต่การวนซ้ำอย่างง่ายผ่านสตริง ไปจนถึงกระบวนการประมวลผลข้อมูลที่ซับซ้อนซึ่งเกี่ยวข้องกับหลายโครงสร้างข้อมูล ความสามารถนี้ในการจัดการกับลำดับและอ็อบเจกต์ที่วนซ้ำได้หลากหลายประเภทช่วยเพิ่มความยืดหยุ่นและประสิทธิภาพให้กับโค้ดของคุณได้อย่างมาก

แนวคิดสำคัญของลูป For:

- วนซ้ำผ่านลำดับข้อมูล เช่น รายการ ทูเพิล ดิกชันนารี เซต หรือสตริง
- เหมาะกับสถานการณ์ที่ทราบจำนวนครั้งในการวนซ้ำล่วงหน้า

```
for variable in sequence:
    # code to execute
```

Listing 4.1: For Loops Syntax

4.2.1 การวนซ้ำผ่านรายการ (List)

การวนซ้ำผ่านรายการ (list) หมายถึงการใช้ลูปเพื่อเข้าถึงและจัดการกับแต่ละองค์ประกอบในรายการตามลำดับ เทคนิคนี้ช่วยให้คุณสามารถดำเนินการต่าง ๆ เช่น การปรับเปลี่ยนองค์ประกอบ การคำนวณ หรือการดึงข้อมูลจากแต่ละรายการ ซึ่งเป็นสิ่งสำคัญต่อการจัดการและประมวลผลข้อมูลในรูปแบบรายการอย่างมีประสิทธิภาพ

Algorithm 9: อัลกอริทึมสำหรับแสดงชื่อผลไม้

Input: รายการของผลไม้
Output: ชื่อของผลไม้ที่แสดงผลออกมา

```
1 begin
2     fruits = ["apple", "banana", "cherry"];
3     for fruit in fruits do
4         Display fruit;
```

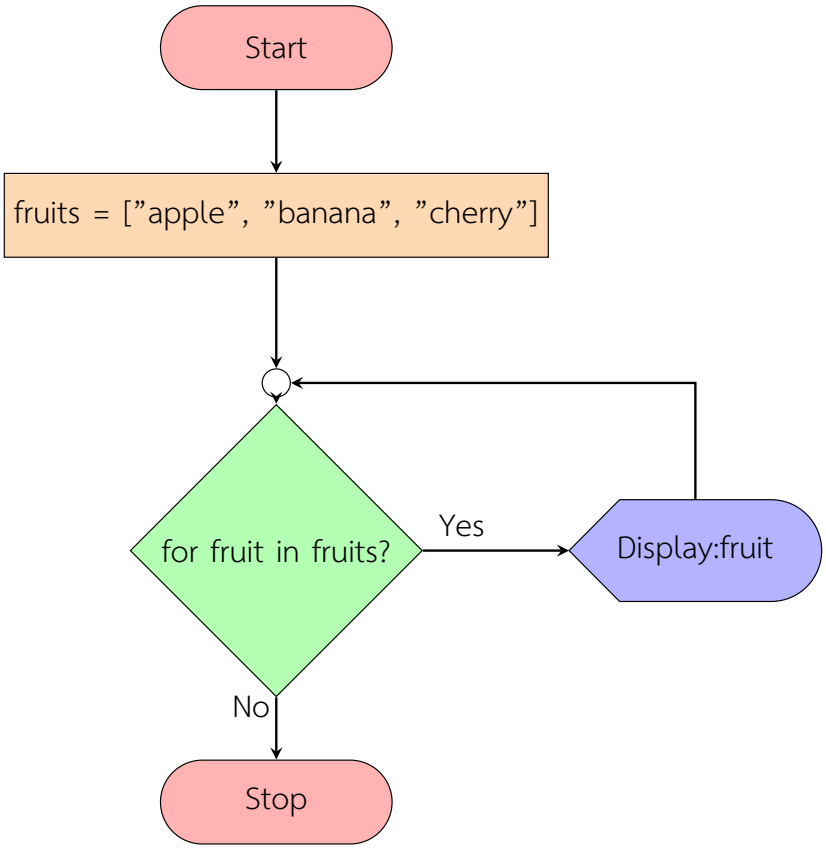

Figure 4.2: ผังงานสำหรับแสดงรายการผลไม้

```python
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Listing 4.2: Example of Iterating Over a List

4.2.2 การวนซ้ำผ่านสตริง

การวนซ้ำผ่านสตริงหมายถึงการใช้ลูปเพื่อเข้าถึงแต่ละตัวอักษรในสตริงแบบแยกกัน วิธีนี้ช่วยให้สามารถดำเนินการกับแต่ละตัวอักษรได้ เช่น การนับจำนวนครั้งที่เกิด การตรวจสอบตัวอักษรเฉพาะ หรือการแปลงแต่ละตัวอักษร จึงเป็นเทคนิคพื้นฐานสำหรับการจัดการกับสตริง

```
for char in "Hello":
    print(char)
```

Listing 4.3: Example of Iterating Over a String

ตัวอย่างเช่น ปัญหาในการแทนที่สระในสตริงจะต้องประมวลผลข้อความที่ผู้ใช้ป้อนเข้ามา โดยแปลงตัวอักษรทั้งหมดให้เป็นตัวพิมพ์ใหญ่ และแทนที่ตัวสระทั้งหมดด้วยเครื่องหมายดอกจัน (*) ขั้นตอนเริ่มจากการรับอินพุตสตริงจากผู้ใช้ โปรแกรมจะวนซ้ำแต่ละตัวอักษรในสตริงอินพุต แปลงแต่ละตัวอักษรเป็นตัวพิมพ์ใหญ่ จากนั้นตรวจสอบว่าตัวอักษรนั้นเป็นสระหรือไม่ (ไม่ว่าจะเป็นตัวพิมพ์เล็กหรือตัวพิมพ์ใหญ่) หากเป็นสระจะถูกแทนที่ด้วยเครื่องหมายดอกจัน มิฉะนั้นจะเก็บตัวอักษรพิมพ์ใหญ่ไว้ในผลลัพธ์ สุดท้ายจะแสดงผลลัพธ์ของสตริงที่ได้รับการปรับเปลี่ยนแล้ว โดยที่ตัวสระทั้งหมดถูกแทนที่ และตัวอักษรทั้งหมดเป็นตัวพิมพ์ใหญ่ การฝึกฝนนี้ช่วยเสริมทักษะในการจัดการสตริง การตรวจเงื่อนไข และการวนซ้ำในโปรแกรม

Algorithm 10: อัลกอริทึมสำหรับแทนที่สระในสตริงด้วยเครื่องหมายดอกจัน

Input: สตริงจากผู้ใช้
Output: สตริงที่ถูกแทนที่สระด้วยเครื่องหมายดอกจัน

```
1  begin
2      อ่านสตริงจากผู้ใช้;
3      กำหนดสตริงว่างสำหรับเก็บผลลัพธ์;
4      กำหนดตัวแปรเก็บสระเป็น "aeiouAEIOU";
5      for แต่ละตัวอักษรในสตริงอินพุต do
6          แปลงตัวอักษรเป็นตัวพิมพ์ใหญ่;
7          if ตัวอักษรเป็นสระ then
8              เพิ่มเครื่องหมายดอกจันลงในผลลัพธ์;
9          else
10             เพิ่มตัวพิมพ์ใหญ่ลงในผลลัพธ์;
11     แสดงสตริงที่ถูกปรับเปลี่ยนแล้ว;
```

ขั้นตอนการเขียนโปรแกรม อ่านสตริงจากผู้ใช้

- แจ้งให้ผู้ใช้ป้อนสตริงเข้ามา
- เก็บสตริงไว้ในตัวแปร

กำหนดสตริงว่างสำหรับเก็บผลลัพธ์

- สร้างตัวแปรสตริงว่างเพื่อเก็บข้อความที่ถูกปรับเปลี่ยน

กำหนดสระ

- สร้างตัวแปรที่เป็นสตริงเก็บสระทั้งหมดทั้งตัวพิมพ์เล็กและพิมพ์ใหญ่ (”aeiouAEIOU”) เพื่อใช้ตรวจสอบกับแต่ละตัวอักษรในสตริง

วนซ้ำผ่านแต่ละตัวอักษรในสตริงอินพุต

- ใช้ลูป `for` เพื่อวนผ่านแต่ละตัวอักษรในสตริง
- สำหรับแต่ละตัวอักษร:
  - แปลงเป็นตัวพิมพ์ใหญ่
  - ตรวจสอบว่าตัวอักษรเป็นสระโดยดูว่ามีอยู่ในตัวแปรเก็บสระหรือไม่
  - ถ้าเป็นสระให้เพิ่มเครื่องหมายดอกจันลงในสตริงผลลัพธ์
  - ถ้าไม่ใช่สระ ให้เพิ่มตัวพิมพ์ใหญ่ลงในสตริงผลลัพธ์

แสดงผลสตริงที่ปรับเปลี่ยนแล้ว

- เมื่อสิ้นสุดลูป ให้แสดงสตริงที่มีตัวพิมพ์ใหญ่และสระถูกแทนที่ด้วยเครื่องหมายดอกจัน

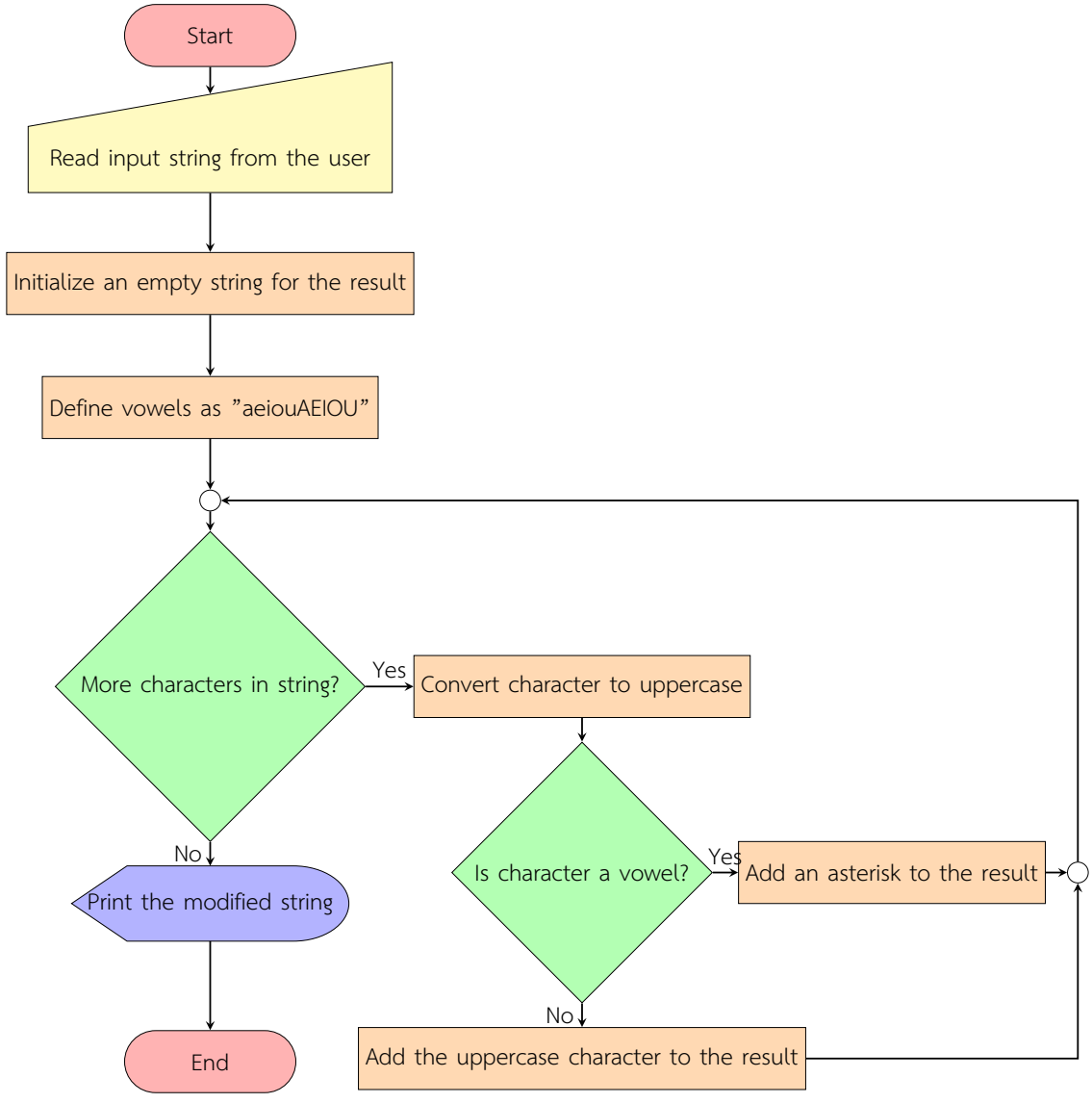

Figure 4.3: ผังงานสำหรับการปรับเปลี่ยนสตริงโดยแทนที่สระด้วยเครื่องหมายดอกจัน

```python
# Input string from the user
input_string = input("Enter a string: ")

# Initialize an empty string for the modified result
modified_string = ""

# Define a set of vowels
vowels = "aeiouAEIOU"

# Use a for loop to iterate through the input string
for char in input_string:
    # Convert character to uppercase
    upper_char = char.upper()
    # Replace vowels with asterisks
    if upper_char in vowels:
        modified_string += "*"
    else:
        modified_string += upper_char

# Print the modified string
print("Modified string:", modified_string)
```

Listing 4.4: Example of String Modification

4.2.3 การใช้ฟังก์ชัน Range

ในลูป `for` การใช้ฟังก์ชัน `range()` ร่วมกับพารามิเตอร์ `start`, `stop` และ `step` ช่วยให้สามารถควบคุมการวนซ้ำผ่านลำดับตัวเลขได้อย่างยืดหยุ่น โดยพารามิเตอร์ `start` ใช้ระบุจุดเริ่มต้นของลำดับ `stop` เป็นค่าที่สิ้นสุด (แต่ไม่รวมค่า) และ `step` ใช้กำหนดระยะห่างระหว่างตัวเลขแต่ละตัว

```python
# Example Syntax
for i in range(start, stop, step):
    #Statement 1...
    #Statement 2...
```

Listing 4.5: Syntax of for-range()

```python
# Print numbers from 0 to 4
for i in range(5):
    print(i)
#This will print the numbers 0, 1, 2, 3, and  4.
```

Listing 4.6: Example of Using the Range Function

```python
# Print numbers from 3 to 9
for i in range(3,10):
    print(i)
#This will print the numbers 3, 4, 5, 6, 7, 8, and 9.
```

Listing 4.7: Example of Using the Range Function with Start and Stop

```python
# Print numbers from 1 to 10 with a step of 2
for i in range(1, 11, 2):
    print(i)
#This will print the numbers 1, 3, 5, 7, and 9.
```

Listing 4.8: Example of Using the Function with Start Stop and Step

สถานการณ์และปัญหาที่ต้องใช้รูปแบบต่าง ๆ ของ `for...range()`

1. การใช้ `for...range(stop)`

สถานการณ์: เมื่อคุณต้องการวนซ้ำจำนวนครั้งที่แน่นอน โดยเริ่มจาก 0 ไปจนถึง (แต่ไม่รวม) ค่าที่กำหนดใน `stop`
ตัวอย่างปัญหา: วนซ้ำผ่านลำดับข้อมูลตามจำนวนที่กำหนดไว้ล่วงหน้า
ตัวอย่างการใช้งาน:

- วนซ้ำผ่านดัชนีของอาเรย์
- ทำซ้ำกระบวนการจำนวนหนึ่งครั้งที่แน่นอน

2. การใช้ `for...range(start, stop)`

สถานการณ์: เมื่อคุณต้องการวนซ้ำจากค่าที่กำหนดจนถึง (แต่ไม่รวม) ค่าที่สิ้นสุด
ตัวอย่างปัญหา: วนซ้ำผ่านบางส่วนของข้อมูลหรือทำงานภายในช่วงค่าที่กำหนด
ตัวอย่างการใช้งาน:

- สร้างลำดับตัวเลขภายในช่วงที่กำหนด
- ทำงานกับบางส่วนของโครงสร้างข้อมูล

3. การใช้ `for...range(start, stop, step)`

สถานการณ์: เมื่อคุณต้องการวนซ้ำจากค่าหนึ่งไปยังอีกค่าหนึ่ง โดยเว้นระยะห่างตามค่าที่กำหนดไว้ใน `step`
ตัวอย่างปัญหา: วนซ้ำแบบเว้นระยะ หรือย้อนลำดับ
ตัวอย่างการใช้งาน:

- ข้ามบางองค์ประกอบในลำดับ
- วนซ้ำแบบย้อนลำดับ
- สร้างลำดับที่ไม่เป็นเชิงเส้น

สรุป

- `for...range(stop)`: เหมาะสำหรับการวนซ้ำแบบพื้นฐานที่เริ่มจากศูนย์
- `for...range(start, stop)`: เหมาะสำหรับการวนซ้ำภายในช่วงค่าที่กำหนด
- `for...range(start, stop, step)`: ใช้ได้ดีสำหรับการวนซ้ำที่ต้องการรูปแบบเฉพาะ เช่น การข้ามค่า หรือวนซ้ำแบบย้อนกลับ

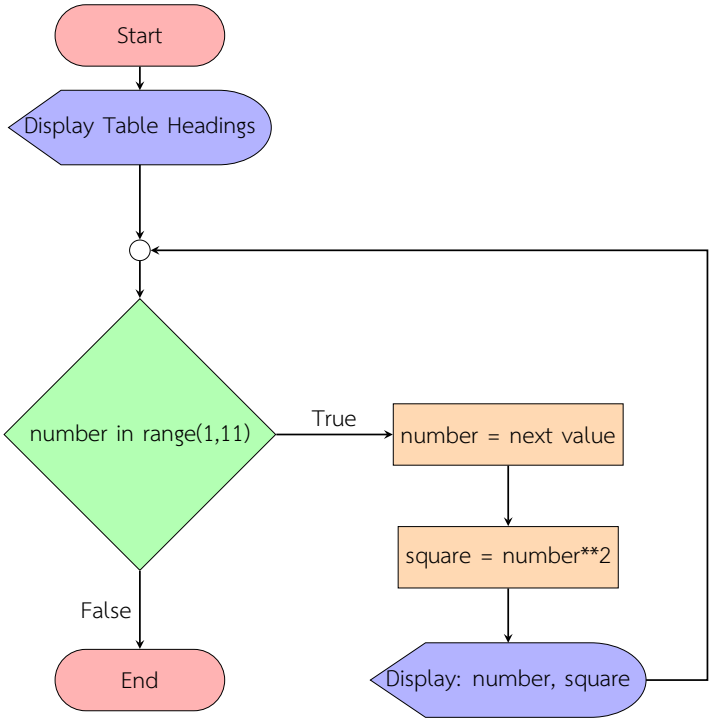

Figure 4.4: การใช้ for..range() เพื่อประมวลผลรายการของค่า

4.3 ลูป While

ลูป `while` จะทำการประมวลผลบล็อกของโค้ดซ้ำไปเรื่อย ๆ ตราบใดที่เงื่อนไขที่กำหนดยังคงเป็นจริง ซึ่งหมายความว่าเงื่อนไขจะถูกตรวจสอบก่อนการทำงานแต่ละครั้ง และหากเป็นจริง โค้ดภายในลูปจะถูกดำเนินการ วนซ้ำไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ ลูปประเภทนี้เหมาะกับสถานการณ์ที่ไม่สามารถกำหนดจำนวนรอบในการทำงานได้ล่วงหน้า และต้องพึ่งพาปัจจัยแบบไดนามิก เช่น การป้อนข้อมูลจากผู้ใช้ หรือผลลัพธ์จากการคำนวณภายในลูป

ตัวอย่างการใช้งาน เช่น การอ่านข้อมูลจากไฟล์จนกว่าจะถึงจุดสิ้นสุดของไฟล์ หรือการถามผู้ใช้ซ้ำ ๆ จนกว่าจะได้รับคำตอบที่ถูกต้อง เนื่องจากการทำงานของลูปขึ้นอยู่กับเงื่อนไข ลูปจึงเป็นเครื่องมือที่ทรงพลังสำหรับการทำงานซ้ำที่ต้องการโครงสร้างควบคุมที่ยืดหยุ่นและตอบสนองได้ อย่างไรก็ตาม การรับประกันว่าเงื่อนไขจะกลายเป็นเท็จในบางจุดเป็นสิ่งสำคัญ เพื่อหลีกเลี่ยงลูปที่ไม่มีวันสิ้นสุด ซึ่งจะทำให้โปรแกรมทำงานไม่หยุดหย่อน หากใช้อย่างถูกต้อง ลูปถือเป็นเครื่องมือสำคัญสำหรับจัดการกระบวนการที่ซับซ้อนในการเขียนโปรแกรม

แนวคิดสำคัญของลูป While:

- ดำเนินการซ้ำตราบใดที่เงื่อนไขที่กำหนดยังคงเป็นจริง
- เหมาะสำหรับสถานการณ์ที่ไม่สามารถทราบจำนวนรอบล่วงหน้า

```
while condition:
    # code to execute
```

Listing 4.9: While Loops Syntax

4.3.1 ลูป While และเงื่อนไข

ลูป `while` ที่มีเงื่อนไขจบการทำงานจะทำซ้ำโค้ดตราบใดที่เงื่อนไขยังเป็นจริง ก่อนแต่ละรอบจะมีการตรวจสอบเงื่อนไข และลูปจะดำเนินต่อไปจนกว่าเงื่อนไขจะกลายเป็นเท็จ ลูปประเภทนี้เหมาะกับกรณีที่ไม่ทราบจำนวนรอบแน่นอนล่วงหน้า และขึ้นอยู่กับปัจจัยแบบไดนามิก เช่น การป้อนข้อมูลจากผู้ใช้ หรือผลลัพธ์จากการคำนวณ การรับประกันว่าเงื่อนไขจะเป็นเท็จในบางจุดเป็นสิ่งสำคัญเพื่อหลีกเลี่ยงลูปไม่มีที่สิ้นสุด การจัดการเงื่อนไขจบลูปอย่างเหมาะสมจะช่วยให้ลูปทำงานได้ตามวัตถุประสงค์ และทำให้โปรแกรมดำเนินการหรือสิ้นสุดได้อย่างถูกต้อง

```
count = 0
while count < 5:
    print("Hello : ", count)
    count += 1
```

Listing 4.10: Example of While Loops with End Condition

4.3.2 ลูปไม่มีที่สิ้นสุด (Infinite Loops)

ควรระมัดระวังในการใช้ลูป `while` เพื่อให้แน่ใจว่าเงื่อนไขจะกลายเป็นเท็จในบางจุด มิฉะนั้นอาจเกิดลูปไม่มีที่สิ้นสุด ซึ่งจะทำให้โปรแกรมทำงานไม่หยุด ส่งผลให้ไม่สามารถตอบสนองได้และใช้ทรัพยากรมากเกินไป วิธีหลีกเลี่ยงลูปประเภทนี้คือการทำให้เงื่อนไขภายในลูปมีการเปลี่ยนแปลง เช่น การปรับค่าตัวแปร หรือการใช้คำสั่ง `break` นอกจากนี้ การเพิ่มตัวควบคุมจำนวนรอบสูงสุดก็เป็นวิธีหนึ่งในการป้องกันการทำงานแบบไม่รู้จบ การจัดการเงื่อนไขของลูปอย่างเหมาะสมและการเพิ่มกลไกป้องกันช่วยให้ลูปสามารถทำงานได้อย่างปลอดภัยและมีประสิทธิภาพ

```
# This loop will run forever
while True:
    print("This is an infinite loop.")
```

Listing 4.11: Example of Infinite Loops

4.3.3 เซนทิเนล (Sentinel)

การใช้เซนทิเนลร่วมกับลูป `while` เป็นรูปแบบการเขียนโปรแกรมที่ควบคุมการทำงานของลูปโดยใช้ค่าหรือเงื่อนไขพิเศษเป็นตัวสิ้นสุดลูป เซนทิเนลมีประโยชน์มากในกรณีที่จำนวนรอบของลูปขึ้นอยู่กับอินพุตแบบไดนามิก เช่น การรับข้อมูลจากผู้ใช้ซ้ำ ๆ จนกว่าจะป้อนคำเฉพาะ (เช่น `'exit'`) รูปแบบนี้ช่วยให้การวนซ้ำมีความยืดหยุ่นและสามารถควบคุมได้ดี ซึ่งเหมาะอย่างยิ่งกับสถานการณ์ที่ข้อมูลมีความไม่แน่นอน

```
# This program calculates sales commissions.
# Create a variable to control the loop.
keep_going = 'y'

# Calculate a series of commissions.
while keep_going == 'y':
    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))

    # Calculate the commission.
    commission = sales * comm_rate

    # Display the commission.
    print(f'The commission is ${commission:.2f}')

    # See if the user wants to do another one.
    keep_going = input('Do you want to calculate another' + \
                       ' commission (Enter y for yes): ')
```

Listing 4.12: Program to calculate sales commissions

Algorithm 11: แบบฝึกหัด: อัลกอริทึมทายหมายเลขมหัศจรรย์

Input: ค่าที่ผู้ใช้ทาย (ระหว่าง 1 ถึง 100)
Output: ข้อความบอกว่าทายถูกหรือไม่

```
begin
    import random
    แสดงข้อความ "What is my magic number (1 to 100)?";
    สุ่มหมายเลขระหว่าง 1 ถึง 100 และเก็บไว้ใน mynumber;
    กำหนดค่าเริ่มต้น ntries เป็น 1;
    กำหนดค่าเริ่มต้น yourguess เป็น -1;
    while ntries < 7 และ yourguess ≠ mynumber do
        กำหนดข้อความ msg ให้เท่ากับ ntries + ">> ";
        if ntries == 6 then
            แสดงข้อความ "Your last chance";
        รับค่าที่ผู้ใช้ทาย และเก็บใน yourguess;
        if yourguess > mynumber then
            แสดงข้อความ "-- > too high";
        else
            แสดงข้อความ "-- > too low";
        เพิ่มค่า ntries ทีละ 1;
    if yourguess == mynumber then
        แสดงข้อความ "Yes! it's", mynumber;
    else
        แสดงข้อความ "Sorry! my number is", mynumber;
```

4.3.4 ลูปตรวจสอบความถูกต้องของข้อมูลที่รับเข้า (Input Validation Loop)

แนวคิดของลูปตรวจสอบความถูกต้องของข้อมูลที่รับเข้า (Input Validation Loop) คือการถามข้อมูลจากผู้ใช้ซ้ำ ๆ จนกว่าจะได้รับข้อมูลที่ถูกต้องตามเงื่อนไขที่กำหนด วิธีนี้ช่วยให้โปรแกรมสามารถจัดการกับข้อมูลที่รับเข้าจากผู้ใช้ได้อย่างมั่นคง ป้องกันข้อผิดพลาด และรักษาความถูกต้องของข้อมูล โดยลูปจะตรวจสอบเงื่อนไข เช่น ช่วงของค่า ประเภทข้อมูล หรือรูปแบบที่ต้องการ หากข้อมูลที่ผู้ใช้ป้อนไม่ตรงตามเงื่อนไข ระบบจะแสดงข้อความแสดงข้อผิดพลาด และถามข้อมูลใหม่จนกว่าจะได้รับข้อมูลที่ถูกต้อง ลูปประเภทนี้มีความสำคัญต่อการสร้างโปรแกรมที่ใช้งานง่ายและทนทานต่อข้อมูลผิดพลาดจากผู้ใช้

```
# Get a test score.
score = int(input('Enter a test score: '))

# Make sure it is not less than 0 or greater than 100.
while score < 0 or score > 100:
    print('ERROR: The score cannot be negative')
    print('or greater than 100.')
    score = int(input('Enter the correct score: '))
```

Listing 4.13: Program to get a test score and validate it

4.4 คำสั่งควบคุมลูป (Loop Control Statements)

คำสั่งควบคุมลูป เช่น `break`, `continue`, และ `pass` ใช้เพื่อควบคุมการทำงานของลูปในโปรแกรม คำสั่ง `break` ใช้สำหรับออกจากลูปทันทีเมื่อเงื่อนไขที่กำหนดเป็นจริง ซึ่งช่วยให้ลูปหยุดก่อนจะทำงานครบทุกครั้งตามรอบปกติ เพื่อเพิ่มประสิทธิภาพหรือหยุดเมื่อได้ผลลัพธ์ที่ต้องการ คำสั่ง `continue` ใช้เพื่อข้ามการทำงานในรอบนั้นของลูป และไปทำรอบถัดไปทันที โดยไม่หยุดลูปทั้งหมด ส่วนคำสั่ง `pass` เป็นคำสั่งว่าง ใช้เป็นตัวแทนในกรณียังไม่ได้เขียนโค้ดหรือใช้เพื่อให้โครงสร้างโปรแกรมสมบูรณ์ตามไวยากรณ์ คำสั่งเหล่านี้ช่วยเพิ่มความยืดหยุ่นและประสิทธิภาพให้กับลูป ช่วยให้สามารถควบคุมการทำงานของโค้ดภายในลูปได้อย่างแม่นยำ

แนวคิดสำคัญของคำสั่งควบคุมลูป:

- **Break**: ออกจากลูปทันทีเมื่อเงื่อนไขที่กำหนดเป็นจริง
- **Continue**: ข้ามรอบปัจจุบันของลูปและไปยังรอบถัดไปทันที
- **Pass**: ไม่มีการดำเนินการใด ๆ ใช้เป็นตัวแทนโค้ดที่ยังไม่ถูกเขียน

4.4.1 คำสั่ง Break

คำสั่ง `break` ใช้เพื่อออกจากลูปทันทีเมื่อเงื่อนไขที่กำหนดเป็นจริง วิธีนี้ช่วยหยุดการทำงานของลูปก่อนที่จะวนซ้ำครบทั้งหมด ซึ่งช่วยเพิ่มประสิทธิภาพของโปรแกรมหรือป้องกันการทำงานที่ไม่จำเป็นเมื่อได้ผลลัพธ์ที่ต้องการแล้ว

```
for i in range(10):
    if i == 5:
        break
    print(i)  # Output: 0 1 2 3 4
```

Listing 4.14: Example of Break Statement

4.4.2 คำสั่ง Continue

คำสั่ง `continue` ใช้เพื่อข้ามการทำงานในรอบปัจจุบันของลูป และไปยังรอบถัดไปทันที ซึ่งหมายความว่าเมื่อเงื่อนไขที่กำหนดเป็นจริง ลูปจะข้ามส่วนของโค้ดที่เหลือในรอบนั้นและเริ่มการวนซ้ำรอบใหม่ทันที วิธีนี้ช่วยให้สามารถควบคุมการทำงานเฉพาะบางรอบของลูปได้อย่างยืดหยุ่น

```
for i in range(10):
    if i == 5:
        continue
    print(i)  # Output: 0 1 2 3 4 6 7 8 9
```

Listing 4.15: Example of Break Statement

4.4.3 คำสั่ง Pass

คำสั่ง `pass` เป็นคำสั่งที่ไม่มีการกระทำใด ๆ (null operation) ใช้ในกรณีที่ต้องมีคำสั่งในทางไวยากรณ์ (syntactically required) แต่ยังไม่ต้องการให้โค้ดใด ๆ ทำงาน คำสั่งนี้มีประโยชน์เมื่อคุณต้องการเว้นที่ไว้ในลูปหรือโครงสร้างเงื่อนไขที่ยังไม่ต้องการดำเนินการในขณะนั้น เช่น ระหว่างการพัฒนาโปรแกรม

```
for i in range(5):
    if i == 3:
        pass
    else:
        print(i)  # Output: 0 1 2 4
```

Listing 4.16: Example of Break Statement

4.5 การคำนวณผลรวมสะสม (Running Total)

การคำนวณผลรวมสะสมหมายถึงการบวกค่าตัวเลขในลำดับอย่างต่อเนื่องในขณะที่ประมวลผลข้อมูล เทคนิคนี้มีประโยชน์อย่างมากในหลายบริบท เช่น การคำนวณทางการเงิน การวิเคราะห์ข้อมูล และระบบติดตามแบบเรียลไทม์ ตัวอย่างเช่น ผลรวมสะสมในบริบททางเศรษฐกิจช่วยในการติดตามยอดรวมของธุรกรรมตลอดช่วงเวลา ในการวิเคราะห์ข้อมูล เทคนิคนี้ช่วยให้เข้าใจแนวโน้มและการเปลี่ยนแปลงในชุดข้อมูลได้ดีขึ้น

การคำนวณผลรวมสะสมจะทำโดยเริ่มจากการกำหนดตัวแปรสำหรับเก็บค่าผลรวม จากนั้นบวกค่าตัวเลขแต่ละค่าจากลำดับเข้าไปยังตัวแปรนี้อย่างต่อเนื่อง วิธีนี้ช่วยให้สามารถติดตามค่ารวมที่เปลี่ยนแปลงได้อย่างมีประสิทธิภาพเมื่อข้อมูลใหม่ถูกประมวลผล

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# Variable to store the sum
sum = 0

# Iterate over the list
for val in numbers:
    sum += val
    print(sum)

print("The sum is", sum)
```

Listing 4.17: Calculates the sum of a series of numbers entered by the user

```
# This program calculates the sum of a series of numbers the user enters.

max = 5  # The maximum number
# Initialize an accumulator variable.
total = 0.0

# Explain what we are doing.
print('This program calculates the sum of')
print(max, 'numbers you will enter.')

# Get the numbers and accumulate them.
for counter in range(max):
    number = int(input('Enter a number: '))
    total = total + number

# Display the total of the numbers.
print('The total is', total)
```

Listing 4.18: Calculates the sum of a series of numbers entered by the user

4.6 ลูปซ้อน (Nested Loop)

ลูป `for` ซ้อนกัน (Nested-for loops) หมายถึงการเขียนลูปหนึ่งไว้ภายในอีกลูปหนึ่ง ซึ่งช่วยให้สามารถวนซ้ำผ่านโครงสร้างข้อมูลหลายมิติ หรือดำเนินการซ้ำภายในแต่ละรอบของลูปรอบนอกได้ ลูปด้านในจะทำงานครบทุกครั้งก่อนที่ลูปรอบนอกจะขยับไปยังรอบถัดไป เทคนิคนี้มีประโยชน์อย่างมากในการทำงานกับตาราง การสร้างชุดค่าผสม หรือการประมวลผลข้อมูลที่ซับซ้อน เช่น การเข้าถึงแต่ละองค์ประกอบในลิสต์สองมิติ (เมทริกซ์) โดยใช้ลูปรอบนอกสำหรับแถว และลูปรอบในสำหรับคอลัมน์ การจัดการลูปซ้อนอย่างเหมาะสมเป็นสิ่งสำคัญเพื่อหลีกเลี่ยงความซับซ้อนของการคำนวณที่มากเกินไป และเพื่อให้โค้ดทำงานได้อย่างมีประสิทธิภาพ

```
# Nested loop to print numbers both using range()
for i in range(1, 3):
    for j in range(2,5):
        print(i,j)

#This will print the numbers 1 2, 1 3, 1 4, 2 2, 2 3 and 2 4.
```

Listing 4.19: Example of Using range() Nested Loop

```
# Nested loop to print numbers using i as an index for the second loop
for i in range(4):
    for j in range(i):
        print(i,j)

#This will print the numbers 1 0, 2 0, 2 1, 3 0, 3 1 and 3 2.
```

Listing 4.20: Example of Using index of outer loop

4.7 ตัวอย่างการใช้งานจริง

ในบทนี้ ตัวอย่างการใช้งานจริงจะแสดงให้เห็นถึงการนำลูปไปประยุกต์ใช้ในสถานการณ์ต่าง ๆ เช่น การคำนวณผลรวมของจำนวนเต็มธรรมชาติ n ตัวแรก เพื่อแสดงให้เห็นถึงความสามารถของลูปในการจัดการกับงานคำนวณที่เกิดซ้ำ

อีกตัวอย่างหนึ่งคือการหาค่าแฟกทอเรียลของตัวเลข ซึ่งเน้นให้เห็นถึงกระบวนการวนซ้ำแบบทวีคูณ นอกจากนี้ยังมีตัวอย่างลูปซ้อนที่ใช้สำหรับวนซ้ำผ่านหลายมิติ เช่น ตาราง เพื่อจัดการกับโครงสร้างข้อมูลที่ซับซ้อนได้อย่างเป็นระบบ

4.7.1 ลูปซ้อน (Nested Loops)

สามารถใช้ลูปซ้อนกันได้ โดยมีลูปหนึ่งทำงานภายในอีกลูปหนึ่ง ลูปด้านในจะทำงานครบทุกครั้งสำหรับแต่ละรอบของลูปรอบนอก ซึ่งเหมาะสำหรับงานที่มีความซับซ้อนมากขึ้น เช่น การประมวลผลอาร์เรย์หลายมิติ หรือการสร้างชุดค่าผสมจากหลายลำดับ

```
for i in range(3):
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

Listing 4.21: Example of Nested Loops

4.7.2 การวนซ้ำผ่าน Dictionary

สามารถใช้ลูปในการวนซ้ำผ่านคีย์ ค่าของ Dictionary หรือทั้งคีย์และค่าพร้อมกัน เพื่อให้เข้าถึงและจัดการข้อมูลภายในได้อย่างมีประสิทธิภาพ การวนผ่านคีย์ช่วยให้สามารถตรวจสอบหรือประมวลผลโดยอิงจากชื่อคีย์ ส่วนการวนค่าช่วยให้จัดการกับข้อมูลได้โดยตรง และการวนคู่คีย์-ค่า (key-value pair) ช่วยให้สามารถปรับเปลี่ยนหรือประมวลผลข้อมูลอย่างครอบคลุม

```
student = {"name": "Alice", "age": 25, "grade": "A"}

# Iterating through keys
for key in student:
    print(key)

# Iterating through values
for value in student.values():
    print(value)

# Iterating through key-value pairs
for key, value in student.items():
    print(f"{key}: {value}")
```

Listing 4.22: Looping Through a Dictionary

4.7.3 ผลรวมของจำนวนเต็มธรรมชาติ n ตัวแรก

การใช้ลูปในการคำนวณผลรวมของจำนวนเต็มธรรมชาติ n ตัวแรกทำได้โดยการวนจาก 1 ถึง n แล้วสะสมค่าทั้งหมดในแต่ละรอบ วิธีนี้แสดงให้เห็นถึงความสามารถของลูปในการทำงานทางคณิตศาสตร์แบบเกิดซ้ำได้อย่างมีประสิทธิภาพ

```
n = int(input("Enter a positive integer: "))
sum = 0
for i in range(1, n + 1):
    sum += i
print(f"The sum of the first {n} natural numbers is {sum}.")
```

Listing 4.23: Example of Sum of First N Natural Numbers

4.7.4 การคำนวณแฟกทอเรียล (Factorial)

การใช้ลูปเพื่อหาค่าแฟกทอเรียลของตัวเลขหมายถึงการวนตั้งแต่ 1 ถึงตัวเลขนั้น และคูณค่าผลรวมเดิมด้วยค่าจากรอบปัจจุบัน วิธีนี้ช่วยให้สามารถหาผลคูณสะสมของจำนวนเต็มทั้งหมดจนถึงค่าที่ระบุ ซึ่งแสดงให้เห็นถึงพลังของลูปในการจัดการงานคำนวณที่เกิดซ้ำและซับซ้อนได้อย่างมีประสิทธิภาพ

```
n = int(input("Enter a positive integer: "))
factorial = 1
for i in range(1, n + 1):
    factorial *= i
print(f"The factorial of {n} is {factorial}.")
```

Listing 4.24: Example of Factorial Calculation

บทที่ 4 โจทย์และแบบฝึกหัด: การใช้ลูปในภาษาไพธอน

4.1 การเริ่มต้นใช้งานลูป
เขียนลูปที่พิมพ์ตัวเลขตั้งแต่ 1 ถึง 5

4.2 การวนซ้ำผ่านรายการ (List)
กำหนดรายการ `fruits = ["apple", "banana", "cherry"]` *เขียนลูปเพื่อพิมพ์ชื่อผลไม้แต่ละชนิด*

4.3 การวนซ้ำผ่านข้อความ (String)
เขียนลูปเพื่อพิมพ์ตัวอักษรแต่ละตัวในสตริง "Python"

4.4 การใช้ฟังก์ชัน `range()`
เขียนลูปที่พิมพ์เลขคู่ตั้งแต่ 2 ถึง 10 โดยใช้ฟังก์ชัน `range()`

4.5 พื้นฐานของลูป `while`
เขียนลูป `while` *ที่พิมพ์ตัวเลขตั้งแต่ 0 ถึง 4*

4.6 การหลีกเลี่ยงลูปไม่มีที่สิ้นสุด
เขียนลูป `while` *ที่พิมพ์ข้อความ "Looping..." จำนวน 3 ครั้ง แล้วหยุด*

4.7 คำสั่ง `break`
เขียนลูปที่พิมพ์ตัวเลขตั้งแต่ 0 ถึง 9 แต่หยุดทันทีเมื่อถึงเลข 7

4.8 คำสั่ง `continue`
เขียนลูปที่พิมพ์ตัวเลขตั้งแต่ 0 ถึง 9 โดยข้ามเลข 5

4.9 คำสั่ง `pass`
เขียนลูปที่วนตั้งแต่ 0 ถึง 4 และใช้คำสั่ง `pass` *เมื่อค่าคือ 2 ส่วนค่าที่เหลือให้พิมพ์ออกมา*

4.10 ลูปซ้อน (Nested Loops)
เขียนลูปซ้อนเพื่อพิมพ์ค่าพิกัดในตารางขนาด 2x3 เช่น (0,0), (0,1), ..., (1,2)

ภาคผนวก

ภาคผนวกนี้ประกอบด้วยแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งานจริง เพื่อเสริมความเข้าใจในเนื้อหาที่กล่าวถึงในบทที่ 4

A4.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Python Crash Course* โดย Eric Matthes
- *Python for Data Analysis* โดย Wes McKinney

บทเรียนออนไลน์:

- Python For Loops - W3Schools
- Python While Loops - GeeksforGeeks

คอร์สเรียนออนไลน์:

- Coursera: Python Data Structures โดย University of Michigan
- Udemy: Python for Data Science and Machine Learning Bootcamp

A4.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). เอกสาร Python - การใช้ลูป. สืบค้นจาก https://docs.python.org/3/tutorial/controlflow.html#for-statements
- Lutz, M. (2013). *Learning Python* (ฉบับที่ 5). O'Reilly Media.

A4.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: ลูป `for` เบื้องต้น

- เขียนโปรแกรม Python เพื่อพิมพ์ตัวเลขตั้งแต่ 1 ถึง 10 โดยใช้ `for loop`
- สร้างรายการชื่อหนังสือที่คุณชอบ แล้วใช้ลูป `for` เพื่อพิมพ์ชื่อหนังสือแต่ละเล่ม

แบบฝึกหัดที่ 2: ลูป `while` เบื้องต้น

- เขียนโปรแกรม Python เพื่อพิมพ์เลขธรรมชาติ 10 ตัวแรก โดยใช้ลูป `while`
- สร้างโปรแกรมที่รับค่าตัวเลขจากผู้ใช้ แล้วพิมพ์ค่าตั้งแต่เลขนั้นลงไปจนถึง 1 โดยใช้ลูป `while`

แบบฝึกหัดที่ 3: ลูปขั้นสูง

- เขียนสคริปต์ Python เพื่อหาค่าแฟกทอเรียลของตัวเลขที่ผู้ใช้ป้อน โดยใช้ `while loop`
- เขียนโปรแกรมที่พิมพ์ลำดับ Fibonacci จนถึงค่าที่ผู้ใช้กำหนด โดยใช้ `for loop`

แบบฝึกหัดที่ 4: คำสั่งควบคุมลูป

- เขียนโปรแกรม Python ที่ใช้ `for loop` พิมพ์เลข 1 ถึง 20 โดยข้ามเลขที่หารด้วย 3 ลงตัว
- เขียนสคริปต์ Python ที่ใช้ `while loop` เพื่อรับค่าจากผู้ใช้ซ้ำ ๆ จนกว่าจะป้อนค่าติดลบ

A4.4 ตัวอย่างการใช้งานจริง

```python
# Python program to print each character of a string using a for loop
string = "Hello, World!"
for char in string:
    print(char)
```

Listing 4.25: Python program to print each character of a string using a for loop

```python
# Python program to sum all numbers from 1 to 100 using a while loop
sum = 0
num = 1
while num <= 100:
    sum += num
    num += 1
print("The sum of numbers from 1 to 100 is:", sum)
```

Listing 4.26: Python program to sum all numbers from 1 to 100 using a while loop

```python
# Python program to print a multiplication table using nested loops
for i in range(1, 11):
    for j in range(1, 11):
        print(f"{i} * {j} = {i * j}")
    print("-" * 15)
```

Listing 4.27: Python program to print a multiplication table using nested loops

```python
# Python script to demonstrate the use of break and continue statements
for num in range(1, 11):
    if num == 5:
        break
    print(num)
print("Loop exited using break statement")

for num in range(1, 11):
    if num == 5:
        continue
    print(num)
print("Skipped number 5 using continue statement")
```

Listing 4.28: Python script to demonstrate the use of break and continue statements

บทที่ 5

ฟังก์ชันและแนวคิดการแบ่งโมดูล

บทนี้นำเสนอภาพรวมอย่างครอบคลุมเกี่ยวกับฟังก์ชันและแนวคิดการแบ่งโมดูลในภาษาไพธอน โดยเน้นแนวคิดสำคัญที่จำเป็นสำหรับการเขียนโปรแกรมอย่างมีประสิทธิภาพ เริ่มจากการแนะนำฟังก์ชัน อธิบายถึงความสำคัญในการจัดระเบียบโค้ด เพิ่มความชัดเจน ส่งเสริมการใช้ซ้ำ และช่วยลดความซับซ้อนในการดีบัก จากนั้นอธิบายไวยากรณ์ของการสร้างฟังก์ชันโดยใช้คำสำคัญ `def` พร้อมรายละเอียดของการใช้งานต่าง ๆ ของฟังก์ชัน

เนื้อหาครอบคลุมการใช้งานพารามิเตอร์ แสดงให้เห็นถึงวิธีการส่งข้อมูลเข้าไปในฟังก์ชัน การใช้หลายพารามิเตอร์และค่าพารามิเตอร์เริ่มต้น นอกจากนี้ยังกล่าวถึงค่าที่ส่งกลับ โดยแสดงให้เห็นว่าฟังก์ชันสามารถคืนค่ากลับไปยังผู้เรียกได้อย่างไร และความสำคัญของคำสั่ง `return` ในการสิ้นสุดการทำงานของฟังก์ชัน

หัวข้อเกี่ยวกับขอบเขตตัวแปร (Scope) ก็มีการกล่าวถึงด้วย โดยอธิบายความแตกต่างระหว่างตัวแปรภายในฟังก์ชัน (local) และตัวแปรภายนอก (global) พร้อมตัวอย่างการใช้งาน `global` เพื่อเข้าถึงหรือปรับค่าตัวแปรจากภายนอก

นอกจากนี้ยังมีตัวอย่างการใช้งานจริงเพื่อเสริมความเข้าใจ เช่น การคำนวณพื้นที่วงกลม การตรวจสอบจำนวนเฉพาะ และการสร้างเครื่องคิดเลขอย่างง่าย ตัวอย่างเหล่านี้เน้นย้ำถึงประโยชน์ของการแบ่งปัญหาที่ซับซ้อนออกเป็นฟังก์ชันย่อยที่จัดการได้ง่าย ช่วยให้จัดการโค้ดได้ดีขึ้นและสนับสนุนแนวทางการเขียนโปรแกรมแบบโมดูลโดยรวม ซึ่งช่วยเสริมสร้างความสามารถในการเขียนโปรแกรมของผู้อ่านได้อย่างมีประสิทธิภาพ

5.1 ฟังก์ชัน (s)

ฟังก์ชันคือกลุ่มของคำสั่งที่มีจุดมุ่งหมายเฉพาะ ซึ่งสามารถเรียกใช้งานได้เมื่อจำเป็น ช่วยจัดระเบียบโค้ดให้เป็นส่วนย่อยที่เข้าใจง่าย เพิ่มความชัดเจน และง่ายต่อการติดตาม ฟังก์ชันยังช่วยให้โค้ดสามารถใช้ซ้ำได้ในหลายส่วนของโปรแกรม โดยไม่ต้องเขียนซ้ำ ซึ่งช่วยประหยัดเวลาและลดความผิดพลาดได้ นอกจากนี้ยังช่วยให้การดีบักง่ายขึ้น เพราะสามารถแยกปัญหาในแต่ละฟังก์ชันได้ชัดเจน การใช้ฟังก์ชันยังช่วยให้โปรแกรมมีโครงสร้างที่ดีขึ้นโดยแบ่งเป็นส่วนย่อยที่สามารถจัดการและปรับปรุงได้ง่าย

แนวคิดสำคัญเกี่ยวกับฟังก์ชัน:

- **การกำหนดฟังก์ชัน**: สร้างฟังก์ชันโดยใช้คำสำคัญ `def` ตามด้วยชื่อฟังก์ชันและพารามิเตอร์
- **พารามิเตอร์**: ตัวแปรที่ใช้รับข้อมูลเข้าในฟังก์ชัน กำหนดไว้ในวงเล็บของหัวฟังก์ชัน
- **ค่าที่ส่งกลับ**: ค่าที่ฟังก์ชันคืนให้ผู้เรียกโดยใช้คำสั่ง `return` ซึ่งจะสิ้นสุดการทำงานของฟังก์ชัน
- **พารามิเตอร์เริ่มต้น**: พารามิเตอร์ที่มีค่าเริ่มต้น เมื่อไม่ได้ส่งค่าใดเข้ามา
- **การเรียกใช้งานฟังก์ชัน**: ใช้ชื่อฟังก์ชันตามด้วยวงเล็บ ซึ่งอาจมีการส่งอาร์กิวเมนต์เข้าไป
- **แนวคิดการแบ่งโมดูล**: แบ่งโปรแกรมออกเป็นฟังก์ชันย่อยเพื่อเพิ่มประสิทธิภาพในการจัดการโค้ดและนำกลับมาใช้ซ้ำ
- **เอกสารประกอบ**: การเพิ่มคำอธิบายหรือ docstring เพื่ออธิบายจุดประสงค์ของฟังก์ชัน พารามิเตอร์ และค่าที่ส่งกลับ เพื่อเพิ่มความเข้าใจและง่ายต่อการดูแลโค้ด

5.1.1 การกำหนดฟังก์ชัน

เราสามารถกำหนดฟังก์ชันโดยใช้คำสำคัญ `def` ตามด้วยชื่อฟังก์ชันและวงเล็บ `()` ซึ่งภายในวงเล็บอาจมีพารามิเตอร์หรือไม่ก็ได้ หัวฟังก์ชันจะตามด้วยเครื่องหมาย `:` และตามด้วยบล็อกของโค้ดที่เยื้องภายในซึ่งเป็นคำสั่งของฟังก์ชัน โครงสร้างนี้ใช้เพื่อกำหนดขอบเขตและวัตถุประสงค์ของฟังก์ชัน

ไวยากรณ์:

```
def function_name(parameters):
    # code block
    return value
```

Listing 5.1: Function Definition

ตัวอย่าง:

```
def greet():
    print("Hello, World!")

# Calling the function
greet()
```

Listing 5.2: Defining a Simple Function

5.1.2 การกำหนดฟังก์ชันอย่างง่าย

การสร้างฟังก์ชันอย่างง่ายที่ไม่มีพารามิเตอร์และไม่ส่งค่ากลับ ใช้คำสำคัญ `def` ตามด้วยชื่อฟังก์ชันและวงเล็บเปล่า `()` ภายในฟังก์ชันให้เขียนคำสั่งที่ต้องการให้ทำงาน ซึ่งสามารถเรียกใช้งานได้โดยไม่ต้องส่งอาร์กิวเมนต์ใด ๆ เข้าไป

ตัวอย่าง:

```
def message():
    print('I am Arthur')
    print('King of the Britons')

# Calling the function
print('I have a message for you.')
message()
print('Goodbye!')
```

Listing 5.3: Defining a Simple Function

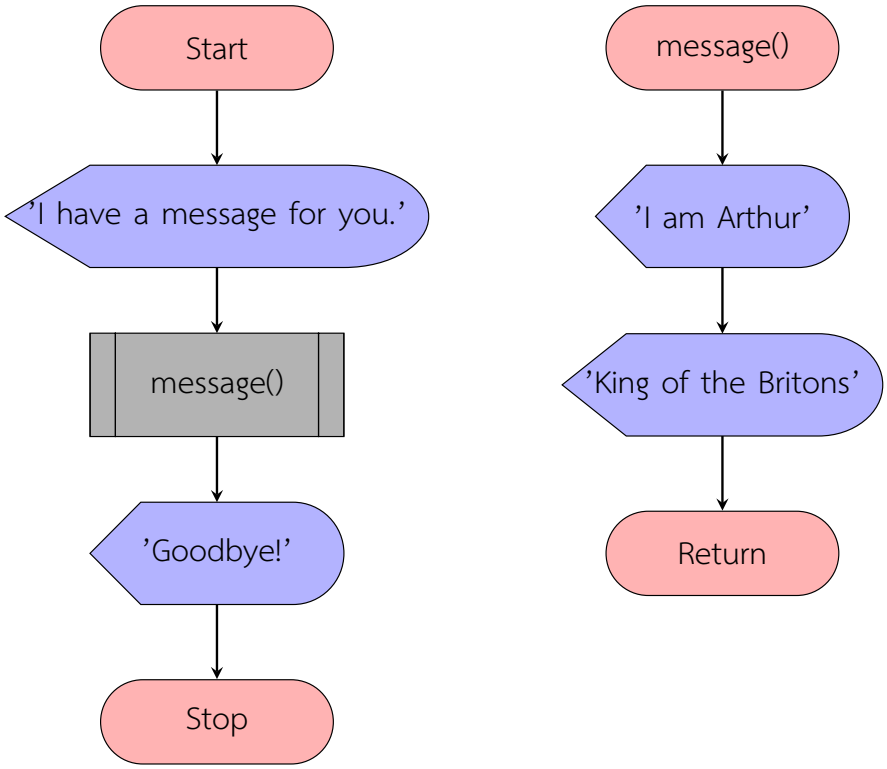

Figure 5.1: ตัวอย่างผังงานของฟังก์ชัน

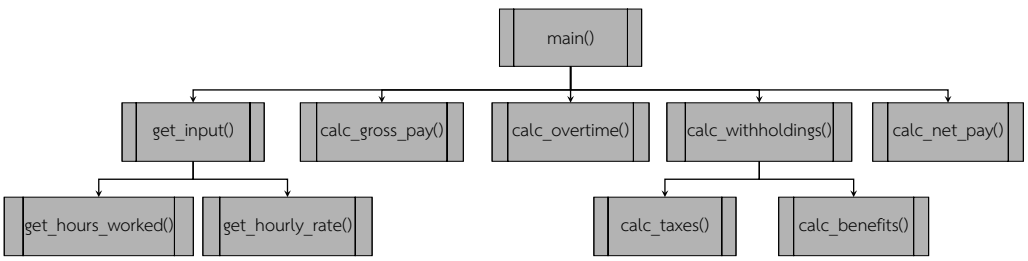

Figure 5.2: ตัวอย่างระบบที่ประกอบด้วยฟังก์ชันย่อย

5.1.3 พารามิเตอร์

ฟังก์ชันที่มีพารามิเตอร์สามารถรับค่าข้อมูลนำเข้าเพื่อใช้ในการประมวลผลตามค่าดังกล่าวได้ พารามิเตอร์จะถูกกำหนดไว้ภายในวงเล็บของหัวฟังก์ชัน เมื่อมีการเรียกใช้ฟังก์ชัน ผู้ใช้ต้องส่งอาร์กิวเมนต์ที่ตรงกับพารามิเตอร์นั้นเข้าไป ซึ่งช่วยให้ฟังก์ชันมีความยืดหยุ่นและสามารถทำงานได้หลากหลายตามข้อมูลนำเข้า

ตัวอย่าง:

```
def greet(name):
    print(f"Hello, {name}!")

# Calling the function with an argument
greet("Alice")
```

Listing 5.4: Function with Parameters

5.1.4 พารามิเตอร์หลายตัว

ฟังก์ชันที่มีพารามิเตอร์หลายตัวสามารถรับค่าข้อมูลนำเข้าหลายค่า ทำให้ฟังก์ชันสามารถทำงานได้อย่างยืดหยุ่นและมีประสิทธิภาพมากขึ้น พารามิเตอร์เหล่านี้จะถูกระบุในวงเล็บของฟังก์ชันโดยคั่นด้วยเครื่องหมายจุลภาค เมื่อเรียกใช้งานฟังก์ชัน ผู้ใช้ต้องส่งอาร์กิวเมนต์ให้ครบถ้วนตามจำนวนพารามิเตอร์ที่กำหนดไว้

ตัวอย่าง:

```
def add(a, b):
    return a + b

# Calling the function with arguments
result = add(3, 5)
print(result)  # Output: 8
```

Listing 5.5: Function with Multiple Parameters

5.1.5 พารามิเตอร์ที่มีค่าเริ่มต้น

ฟังก์ชันที่มีพารามิเตอร์ค่าเริ่มต้นจะกำหนดค่าที่ใช้โดยอัตโนมัติในกรณีที่ไม่มีการส่งอาร์กิวเมนต์สำหรับพารามิเตอร์นั้น ๆ การใช้พารามิเตอร์ค่าเริ่มต้นช่วยให้การเรียกใช้ฟังก์ชันง่ายขึ้นและสามารถปรับเปลี่ยนการทำงานของฟังก์ชันได้ยืดหยุ่นยิ่งขึ้น เหมาะสำหรับกรณีที่บางพารามิเตอร์ไม่จำเป็นต้องระบุทุกครั้ง

ตัวอย่าง:

```
def greet(name="World"):
    print(f"Hello, {name}!")

# Calling the function without an argument
greet()  # Output: Hello, World!

# Calling the function with an argument
greet("Alice")  # Output: Hello, Alice!
```

Listing 5.6: Function with Default Parameters

5.1.6 อาร์กิวเมนต์ที่มีความยาวไม่จำกัด

ภาษา Python อนุญาตให้คุณกำหนดฟังก์ชันที่สามารถรับจำนวนอาร์กิวเมนต์ไม่จำกัดได้ โดยใช้อาร์กิวเมนต์แบบลำดับด้วย `*args` และอาร์กิวเมนต์แบบระบุชื่อด้วย `**kwargs`

ตัวอย่าง:

```
def sum_all(*args):
    return sum(args)

print(sum_all(1, 2, 3, 4, 5))  # Output: 15
```

Listing 5.7: Variable-Length Positional Arguments (*args)

ตัวอย่าง:

```
def find_max(*args):
    if not args:
        return None
    max_value = args[0]
    for number in args:
        if number > max_value:
            max_value = number
    return max_value

# Example usage
result = find_max(3, 5, 7, 2, 8)
print(f"The maximum value is: {result}")  # Output: The maximum value is: 8
```

Listing 5.8: Function to find maximum values

ตัวอย่าง:

```python
def print_all(*args):
    for index, arg in enumerate(args):
        print(f"Argument {index + 1}: {arg}")

# Example usage
print_all("Python", 3.8, True, [1, 2, 3], {"key": "value"})
# Output:
# Argument 1: Python
# Argument 2: 3.8
# Argument 3: True
# Argument 4: [1, 2, 3]
# Argument 5: {'key': 'value'}
```

Listing 5.9: Multiple Type Variable-Length Positional Arguments (*args)

ตัวอย่าง:

```python
def display_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

display_info(name="Alice", age=30, city="New York")
# Output:
# name: Alice
# age: 30
# city: New York
```

Listing 5.10: Variable-Length Keyword Arguments (**kwargs)

ฟังก์ชันที่ใช้อาร์กิวเมนต์เป็นแนวคิดพื้นฐานที่สำคัญในภาษา Python ซึ่งช่วยให้สามารถสร้างโค้ดที่ยืดหยุ่น ใช้งานซ้ำได้ และเป็นโมดูลได้ดี การเข้าใจวิธีการกำหนดและใช้อาร์กิวเมนต์ ทั้งแบบลำดับ แบบกำหนดชื่อ แบบมีค่าเริ่มต้น และแบบไม่จำกัดจำนวน จะช่วยให้คุณสามารถเขียนฟังก์ชันที่มีประสิทธิภาพและเหมาะสมกับความต้องการของโปรแกรมในหลากหลายสถานการณ์ได้

5.1.7 ค่าที่ส่งกลับจากฟังก์ชัน

ฟังก์ชันที่มีการส่งค่ากลับจะใช้คำสั่ง `return` เพื่อส่งผลลัพธ์กลับไปยังผู้เรียกฟังก์ชัน ซึ่งช่วยให้ฟังก์ชันสามารถส่งค่าผลลัพธ์หลังจากทำงานเสร็จสิ้น และสามารถเก็บค่านั้นไว้ในตัวแปรหรือใช้ในกระบวนการอื่น ๆ ต่อไปได้ การใช้ค่าที่ส่งกลับจะเพิ่มความสามารถในการใช้งานของฟังก์ชันให้มีประสิทธิภาพมากขึ้น

ตัวอย่าง:

```
def multiply(a, b):
    return a * b

# Calling the function and storing the returned value
result = multiply(4, 5)
print(result)  # Output: 20
```

Listing 5.11: Function with Return Values

ฟังก์ชัน `calculate_stats` ด้านล่างเป็นฟังก์ชันที่ใช้คำนวณและส่งค่ากลับหลายค่า ได้แก่ ผลรวม ค่าเฉลี่ย ค่าสูงสุด และค่าต่ำสุดของรายการตัวเลข ตัวอย่างการใช้งานแสดงให้เห็นถึงการเรียกใช้ฟังก์ชัน การแตกค่าที่ส่งกลับออกเป็นตัวแปร และการแสดงผลลัพธ์ ซึ่งช่วยให้สามารถคำนวณและเข้าถึงค่าทางสถิติต่าง ๆ ได้อย่างชัดเจนและมีประสิทธิภาพ

ตัวอย่าง:

```
def calculate_stats(numbers):
    total_sum = sum(numbers)
    average = total_sum / len(numbers)
    maximum = max(numbers)
    minimum = min(numbers)
    return total_sum, average, maximum, minimum

# Example usage
numbers = [5, 10, 15, 20, 25]
total, avg, max_num, min_num = calculate_stats(numbers)

print(f"Total Sum: {total}")
print(f"Average: {avg}")
print(f"Maximum: {max_num}")
print(f"Minimum: {min_num}")
```

Listing 5.12: Function with Return Multiple Values

รายละเอียดของฟังก์ชัน `calculate_stats`

โค้ดข้างต้นนิยามฟังก์ชันชื่อ `calculate_stats` ซึ่งใช้ในการคำนวณค่าทางสถิติต่าง ๆ จากรายการตัวเลข ได้แก่ ผลรวม ค่าเฉลี่ย ค่าสูงสุด และค่าต่ำสุด โดยมีรายละเอียดการทำงานดังนี้:

นิยามฟังก์ชัน

```
def calculate_stats(numbers):
    total_sum = sum(numbers)
    average = total_sum / len(numbers)
    maximum = max(numbers)
    minimum = min(numbers)
    return total_sum, average, maximum, minimum
```

- **การประกาศฟังก์ชัน:** ฟังก์ชัน `calculate_stats` มีพารามิเตอร์เดียวชื่อ `numbers` ซึ่งควรเป็นลิสต์ของตัวเลข

คำนวณผลรวม

```
total_sum = sum(numbers)
```

- ใช้ฟังก์ชัน `sum()` เพื่อหาผลรวมของค่าทั้งหมดในลิสต์ `numbers`

คำนวณค่าเฉลี่ย

```
average = total_sum / len(numbers)
```

- หาค่าเฉลี่ยโดยนำผลรวม `total_sum` หารด้วยจำนวนสมาชิกในลิสต์

หาค่าสูงสุด

```
maximum = max(numbers)
```

- ใช้ฟังก์ชัน `max()` เพื่อหาค่าสูงสุดในลิสต์

หาค่าต่ำสุด

```
minimum = min(numbers)
```

- ใช้ฟังก์ชัน `min()` เพื่อหาค่าต่ำสุดในลิสต์

การส่งค่ากลับ

```
return total_sum, average, maximum, minimum
```

- ส่งค่าทั้งสี่ค่ากลับเป็นทูเพิล: `total_sum`, `average`, `maximum`, `minimum`

ตัวอย่างการใช้งาน

```
numbers = [5, 10, 15, 20, 25]
total, avg, max_num, min_num = calculate_stats(numbers)

print(f"Total Sum: {total}")
print(f"Average: {avg}")
print(f"Maximum: {max_num}")
print(f"Minimum: {min_num}")
```

- **รายการตัวเลข:**

```
    numbers = [5, 10, 15, 20, 25]
```

กำหนดลิสต์ของตัวเลข

- **การเรียกใช้ฟังก์ชันและแตกค่า:**

```
    total, avg, max_num, min_num = calculate_stats(numbers)
```

เรียกใช้ฟังก์ชันและรับค่าที่ส่งกลับแยกเป็นตัวแปรต่าง ๆ

- **การแสดงผลลัพธ์:**

```
    print(f"Total Sum: {total}")
    print(f"Average: {avg}")
    print(f"Maximum: {max_num}")
    print(f"Minimum: {min_num}")
```

ผลลัพธ์ที่ได้คือ:

```
Total Sum: 75
Average: 15.0
Maximum: 25
Minimum: 5
```

แบบฝึกหัด

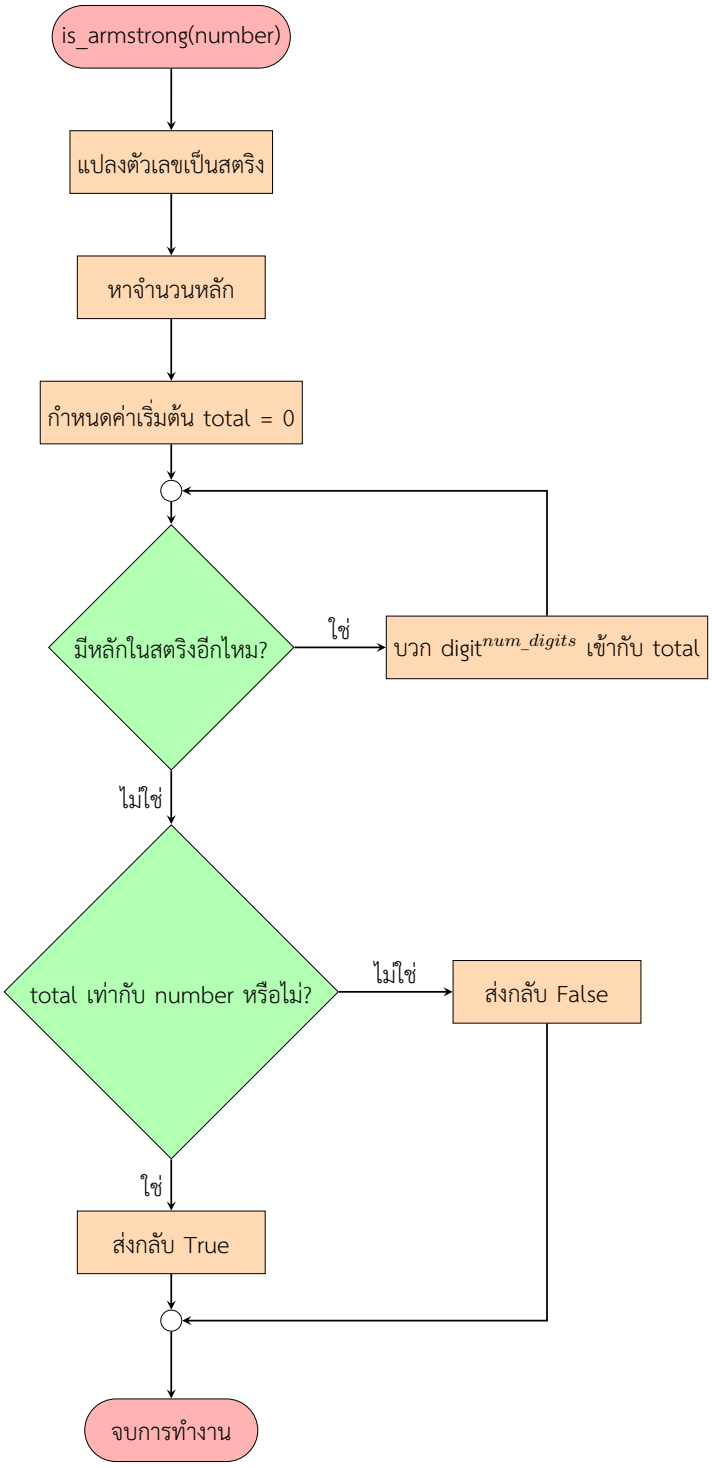

Figure 5.3: ผังงานสำหรับตรวจสอบตัวเลข Armstrong

เลขอาร์มสตรอง (Armstrong Number)

เลขอาร์มสตรอง คือ เลขจำนวนเต็มที่ผลรวมของเลขแต่ละหลักยกกำลังตามจำนวนหลัก เท่ากับตัวเลขนั้นเอง

ตัวอย่าง:

- **153** มี 3 หลัก:

$$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

ดังนั้น 153 เป็นเลขอาร์มสตรอง

- **9474** มี 4 หลัก:

$$9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474$$

ดังนั้น 9474 เป็นเลขอาร์มสตรอง

- **123**:

$$1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36 \neq 123$$

ดังนั้น 123 ไม่ใช่เลขอาร์มสตรอง

ตัวอย่างเลขอาร์มสตรองเพิ่มเติม

1, 153, 370, 371, 407, 9474

5.2 ขอบเขตของฟังก์ชันและตัวแปร (Function Scope and Variables)

ขอบเขตของฟังก์ชันและตัวแปรเป็นแนวคิดสำคัญในภาษาโปรแกรม ขอบเขต (Scope) หมายถึงช่วงที่ตัวแปรสามารถเข้าถึงและทำงานได้ในโปรแกรม ตัวแปรที่ถูกประกาศภายในฟังก์ชันเรียกว่า *ตัวแปรภายใน* ซึ่งสามารถใช้งานได้เฉพาะภายในฟังก์ชันนั้นและจะหมดอายุเมื่อฟังก์ชันจบการทำงาน ในทางกลับกัน *ตัวแปรสากล* ถูกประกาศไว้นอกฟังก์ชันและสามารถเข้าถึงได้จากทุกส่วนของโปรแกรม หากต้องการเปลี่ยนค่าของตัวแปรสากลภายในฟังก์ชัน ต้องใช้คำสั่ง `global` เพื่อระบุอย่างชัดเจน

การเข้าใจขอบเขตของตัวแปรจะช่วยป้องกันความผิดพลาดในการใช้งาน และช่วยให้โค้ดมีโครงสร้างที่ดี ดูแลรักษาได้ง่าย และปลอดภัยมากยิ่งขึ้น

แนวคิดสำคัญเกี่ยวกับขอบเขตของฟังก์ชัน:

- **ขอบเขต (Scope)**: ขอบเขตและช่วงชีวิตของตัวแปร โดยตัวแปรภายในจะถูกจำกัดให้อยู่ภายในฟังก์ชัน ส่วนตัวแปรสากลสามารถเข้าถึงได้จากทุกส่วนของโปรแกรม
- **คำสั่ง `global`**: คำสั่งที่ใช้ในการเปลี่ยนค่าของตัวแปรสากลจากภายในฟังก์ชัน
- **ตัวแปรภายใน**: ตัวแปรที่ถูกประกาศภายในฟังก์ชันและสามารถใช้งานได้เฉพาะภายในฟังก์ชันนั้นเท่านั้น

5.2.1 ขอบเขตของฟังก์ชัน (Function Scope)

ขอบเขตของฟังก์ชันหมายถึงพื้นที่ที่ตัวแปรภายในสามารถมองเห็นและเข้าถึงได้ ตัวแปรที่ประกาศในฟังก์ชันจะใช้งานได้เฉพาะภายในฟังก์ชันนั้น และไม่สามารถเข้าถึงได้จากภายนอก การออกแบบแบบนี้จะช่วยลดความเสี่ยงในการรบกวนค่าของตัวแปรโดยไม่ตั้งใจ และส่งเสริมการเขียนโปรแกรมที่มีระเบียบและปลอดภัยมากยิ่งขึ้น

ตัวอย่าง:

```
def my_function():
    local_variable = "I'm inside the function"
    print(local_variable)

#
my_function()

#         local_variable
# print(local_variable)  # NameError: name 'local_variable' is not defined
```

Listing 5.13: ตัวอย่างการใช้ขอบเขตของฟังก์ชัน

5.2.2 ตัวแปรสากล (Global Variables)

ตัวแปรสากลคือ ตัวแปรที่ถูกประกาศไว้นอกฟังก์ชัน และสามารถเข้าถึงได้จากทุกส่วนของโปรแกรม ตัวแปรเหล่านี้จะคงอยู่ตลอดช่วงการทำงานของโปรแกรม ต่างจากตัวแปรภายในที่หมดอายุเมื่อฟังก์ชันสิ้นสุด ตัวแปรสากลเหมาะกับการใช้ร่วมข้อมูลระหว่างฟังก์ชันต่าง ๆ ในโปรแกรม

ตัวอย่าง:

```
global_variable = "I'm outside the function"

def my_function():
    print(global_variable)

#
my_function()  # Output: I'm outside the function

#
print(global_variable)  # Output: I'm outside the function
```

Listing 5.14: ตัวอย่างการใช้ตัวแปรสากล

5.2.3 การเปลี่ยนค่าตัวแปรสากล (Modifying Global Variables)

หากต้องการเปลี่ยนค่าของตัวแปรสากลจากภายในฟังก์ชัน ต้องใช้คำสั่ง `global` ตามด้วยชื่อตัวแปรนั้น เพื่อบอกให้ฟังก์ชันเข้าถึงตัวแปรสากลแทนที่จะสร้างตัวแปรภายในใหม่ วิธีนี้ทำให้สามารถปรับค่าของตัวแปรสากลได้จากภายในฟังก์ชัน และส่งผลต่อค่าของตัวแปรทั่วทั้งโปรแกรม

ตัวอย่าง:

```
counter = 0

def increment():
    global counter
    counter += 1

#
increment()
increment()

#
print(counter)  # Output: 2
```

Listing 5.15: ตัวอย่างการเปลี่ยนค่าตัวแปรสากล

ตัวอย่าง:

```
#             10
import random

#
HEADS = 1
TAILS = 2
TOSSES = 10

def tosses_coin():
    for toss in range(TOSSES):
        #
        if random.randint(HEADS, TAILS) == HEADS:
            print('Heads')
        else:
            print('Tails')

#
tosses_coin()
```

Listing 5.16: ตัวอย่างการใช้ค่าคงที่และตัวแปรสากลในโปรแกรมโยนเหรียญ

5.3 การจัดเก็บฟังก์ชันในโมดูล (Storing Functions in Modules)

การจัดเก็บฟังก์ชันไว้ในโมดูลเป็นแนวปฏิบัติพื้นฐานที่สำคัญในกระบวนการพัฒนาซอฟต์แวร์ ซึ่งช่วยเพิ่มความเป็นระเบียบ อ่านง่าย ดูแลรักษาได้ง่าย และนำกลับมาใช้ใหม่ได้สะดวก นักพัฒนาสามารถสร้างโครงสร้างโปรแกรมที่เป็นระบบ โดยจัดกลุ่มฟังก์ชันที่เกี่ยวข้องไว้ในโมดูลเดียวกัน ช่วยให้การพัฒนาและดีบักโปรแกรมมีความง่ายและมีประสิทธิภาพมากขึ้น

5.3.1 ความเข้าใจเกี่ยวกับโมดูล (Understanding Modules)

ในภาษา Python โมดูลคือไฟล์ที่ประกอบด้วยโค้ด Python ซึ่งอาจประกอบด้วยการกำหนดฟังก์ชัน คลาส ตัวแปร และโค้ดที่สามารถเรียกใช้งานได้ ข้อดีหลักของการใช้โมดูลคือช่วยแยกโปรแกรมขนาดใหญ่ออกเป็นส่วนย่อย ๆ ที่จัดการได้ง่ายและเป็นระบบ แนวทางแบบโมดูลาร์นี้สอดคล้องกับหลักการ "การแยกหน้าที่ (Separation of Concerns)" ซึ่งหมายถึงการจัดการฟังก์ชันเฉพาะอย่างให้ชัดเจนในแต่ละโมดูล

5.3.2 การสร้างและใช้งานโมดูล (Creating and Using Modules)

การสร้างโมดูลในภาษา Python ทำได้โดยการบันทึกไฟล์ Python ด้วยนามสกุล `.py` ตัวอย่างเช่น ไฟล์ที่ชื่อว่า `math_operations.py` ซึ่งประกอบด้วยฟังก์ชันทางคณิตศาสตร์หลายฟังก์ชัน:

ตัวอย่าง:

```python
# math_operations.py

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Error: Division by zero"
    return a / b
```

Listing 5.17: ตัวอย่างโมดูล math_operations

ไฟล์นี้กลายเป็นโมดูลที่สามารถนำไปใช้ในสคริปต์อื่นได้ โดยสามารถนำฟังก์ชันจาก `math_operations.py` มาใช้งานได้โดยการ import โมดูลในสคริปต์ใหม่:

ตัวอย่าง:

```python
# main.py

import math_operations

result_add = math_operations.add(10, 5)
result_subtract = math_operations.subtract(10, 5)
result_multiply = math_operations.multiply(10, 5)
result_divide = math_operations.divide(10, 5)

print(f"Addition: {result_add}")
print(f"Subtraction: {result_subtract}")
print(f"Multiplication: {result_multiply}")
print(f"Division: {result_divide}")
```

Listing 5.18: ตัวอย่างการนำเข้าและใช้งานโมดูล

5.3.3 การนำเข้าโมดูล (Importing Modules)

มีหลายวิธีในการนำเข้าโมดูลในภาษา Python ได้แก่:

- **นำเข้าโมดูลทั้งชุด:** ตามตัวอย่างข้างต้น ใช้คำสั่ง `import module_name` เพื่อเรียกใช้งานโมดูลทั้งหมด
- **นำเข้าเฉพาะฟังก์ชัน:** สามารถนำเข้าเฉพาะบางฟังก์ชันจากโมดูลได้โดยใช้รูปแบบคำสั่ง `from module_name import function_name` ตัวอย่างเช่น:

ตัวอย่าง:

```
from math_operations import add, subtract

result_add = add(10, 5)
result_subtract = subtract(10, 5)
```

Listing 5.19: การนำเข้าเฉพาะฟังก์ชันจากโมดูล

- **นำเข้าฟังก์ชันทั้งหมด:** ใช้คำสั่ง `from module_name import *` เพื่อนำเข้าทุกฟังก์ชันจากโมดูล อย่างไรก็ตาม วิธีนี้ไม่แนะนำเนื่องจากอาจทำให้ชื่อซ้ำกันใน namespace:

ตัวอย่าง:

```
from math_operations import *

result_add = add(10, 5)
result_subtract = subtract(10, 5)
```

Listing 5.20: การนำเข้าฟังก์ชันทั้งหมดจากโมดูล

- **ใช้ชื่อย่อโมดูล (alias):** สามารถกำหนดชื่อย่อให้กับโมดูลได้โดยใช้รูปแบบ `import module_name as alias` ซึ่งมีประโยชน์เมื่อชื่อโมดูลยาว:

ตัวอย่าง:

```
import math_operations as mo

result_add = mo.add(10, 5)
```

Listing 5.21: การใช้นามแฝง (alias) สำหรับโมดูล

5.3.4 ข้อดีของการใช้โมดูล (Benefits Using Modules)

การใช้โมดูลในภาษา Python เป็นวิธีที่มีประสิทธิภาพในการจัดระเบียบและจัดการโค้ดให้เป็นระบบ โดยการรวมฟังก์ชันและคลาสที่เกี่ยวข้องไว้ในไฟล์แยกกัน โมดูลช่วยให้โค้ดอ่านง่าย ดูแลรักษาง่าย และสามารถขยายระบบได้ง่ายยิ่งขึ้น แนวทางแบบโมดูลยังช่วยให้สามารถนำกลับมาใช้ซ้ำ และสนับสนุนการทำงานร่วมกันของทีมพัฒนาได้อย่างมีประสิทธิภาพ ด้านล่างนี้คือข้อดีที่สำคัญของการใช้โมดูลใน Python:

- **การจัดระเบียบโค้ด:** การจัดเก็บฟังก์ชันในโมดูลช่วยให้โค้ดมีการจัดโครงสร้างอย่างเป็นระเบียบ ตัวอย่างเช่น ฟังก์ชันที่เกี่ยวกับคณิตศาสตร์สามารถจัดรวมไว้ในโมดูลเดียว ส่วนฟังก์ชันที่เกี่ยวกับการจัดการไฟล์อาจแยกไว้ในอีกโมดูลหนึ่ง
- **การนำกลับมาใช้ซ้ำ:** ฟังก์ชันที่อยู่ในโมดูลสามารถนำกลับมาใช้ในโปรแกรมอื่นได้โดยไม่ต้องเขียนซ้ำ ช่วยลดความซ้ำซ้อนของโค้ด
- **การดูแลรักษา:** เมื่อโค้ดถูกแบ่งเป็นโมดูล จะทำให้การแก้ไขบั๊กหรืออัปเดตโค้ดทำได้ง่ายและเฉพาะจุด โปรแกรมที่นำเข้าโมดูลนั้นจะได้รับการอัปเดตโดยอัตโนมัติ
- **การจัดการชื่อ (Namespace):** การใช้โมดูลช่วยป้องกันปัญหาการชนกันของชื่อฟังก์ชันหรือค่าตัวแปร เพราะแต่ละโมดูลจะมี namespace ของตนเอง
- **การทำงานร่วมกันของทีม:** ในงานพัฒนาร่วมกัน โมดูลช่วยให้สมาชิกแต่ละคนสามารถทำงานในส่วนของตนเองได้ และสามารถรวมเข้ากับโครงการหลักได้อย่างง่ายดาย
- **ความสามารถในการขยายระบบ:** เมื่อโปรเจกต์มีขนาดใหญ่ขึ้น การใช้โมดูลจะช่วยให้สามารถเพิ่มฟีเจอร์ใหม่ได้ง่ายโดยไม่รบกวนโค้ดเดิม

5.4 ฟังก์ชันแบบเรียกซ้ำ (Recursive Functions)

ฟังก์ชันแบบเรียกซ้ำ (Recursive Function) คือฟังก์ชันที่เรียกใช้งานตัวเองภายในตัวฟังก์ชันเพื่อลดปัญหาขนาดใหญ่ให้เป็นปัญหาย่อย แนวคิดของการเรียกซ้ำเป็นพื้นฐานที่สำคัญในวิทยาการคอมพิวเตอร์ โดยช่วยให้การแก้ปัญหาซับซ้อนทำได้ง่ายขึ้น เช่น การคำนวณทางคณิตศาสตร์ การจัดการโครงสร้างข้อมูล และการใช้ในอัลกอริธึม

5.4.1 ความเข้าใจเกี่ยวกับการเรียกซ้ำ (Understanding Recursion)

ฟังก์ชันแบบเรียกซ้ำประกอบด้วยสองส่วนหลัก:

Base Case (กรณีพื้นฐาน): เงื่อนไขที่ใช้สิ้นสุดการเรียกซ้ำ เป็นกรณีที่สามารถให้คำตอบได้โดยไม่ต้องเรียกซ้ำอีกต่อไป

Recursive Case (กรณีเรียกซ้ำ): ส่วนที่ฟังก์ชันเรียกตัวเองด้วยอาร์กิวเมนต์ที่เปลี่ยนแปลงเพื่อเข้าใกล้ base case ทีละขั้น

5.4.2 ตัวอย่าง: การคำนวณแฟกทอเรียล (Factorial)

แฟกทอเรียลของจำนวนเต็มบวก n คือผลคูณของจำนวนตั้งแต่ 1 ถึง n นิยามแบบเรียกซ้ำได้ดังนี้:

$$0! = 1 \quad \text{(base case)}$$
$$n! = n \times (n-1)! \quad \text{สำหรับ } n > 0 \quad \text{(recursive case)}$$

ตัวอย่าง:

```
def factorial(n):
    if n == 0:
        return 1  # Base case
    else:
        return n * factorial(n - 1)  # Recursive case

#
print(factorial(5))  # Output: 120
```

Listing 5.22: การคำนวณแฟกทอเรียลแบบเรียกซ้ำ

5.4.3 ข้อดีของฟังก์ชันแบบเรียกซ้ำ (Advantages)

- **เข้าใจง่ายและกระชับ:** ฟังก์ชันแบบเรียกซ้ำสามารถแสดงแนวคิดของปัญหาได้ชัดเจน โดยไม่ต้องใช้ลูปที่ซับซ้อน
- **เหมาะกับโครงสร้างข้อมูล:** เหมาะกับการประมวลผลข้อมูลเชิงโครงสร้าง เช่น ต้นไม้ (tree), กราฟ (graph), และอัลกอริธึมค้นหา
- **โมดูลาร์:** ทำให้โค้ดมีความเป็นโมดูลและเข้าใจได้ง่าย โดยแบ่งปัญหาเป็นส่วนย่อย ๆ

5.4.4 ข้อเสียของฟังก์ชันแบบเรียกซ้ำ (Disadvantages)

- **ประสิทธิภาพ:** มี overhead จากการเรียกฟังก์ชันหลายครั้ง ทำให้ช้ากว่าการใช้ลูปในบางกรณี
- **การใช้หน่วยความจำ:** การเรียกซ้ำหลายชั้นอาจทำให้หน่วยความจำเต็ม (stack overflow)
- **ความซับซ้อนในการดีบัก:** หากไม่มี base case ที่ชัดเจน อาจทำให้เกิดการวนซ้ำไม่รู้จบ

5.4.5 Tail Recursion

Tail recursion คือรูปแบบหนึ่งของ recursion ที่การเรียกตัวเองเป็นคำสั่งสุดท้ายในฟังก์ชัน ช่วยให้สามารถ optimize stack แต่ Python ไม่รองรับการ optimize tail recursion

นิยาม

Tail Recursion คือ ฟังก์ชันที่เรียกตัวเองเป็นคำสั่งสุดท้ายในบรรทัดท้ายสุดของฟังก์ชัน เหมาะสำหรับการ optimize การใช้ stack ในบางภาษา (เช่น Scheme, Haskell)

ตัวอย่างใน Python (Tail Recursion)

```
def tail_factorial(n, acc=1):
    if n == 0:
        return acc
    return tail_factorial(n - 1, acc * n)
```

Listing 5.23: Tail Recursive Function

หมายเหตุ: แม้ฟังก์ชันนี้เป็น tail recursion แต่ Python ไม่รองรับการทำ *Tail Call Optimization (TCO)* หากเรียกซ้ำลึกเกินไป จะเกิด `RecursionError`

ทางเลือก: ใช้ Loop แทน

```python
def factorial_iter(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

Listing 5.24: Iterative Version

สรุป: Python เขียน tail recursion ได้ แต่ไม่ optimize จึงควรใช้ loop เมื่อจำเป็นต้องคำนวณลึก ๆ

5.4.6 ตัวอย่าง: ลำดับฟีโบนัชชี (Fibonacci Sequence)

ลำดับฟีโบนัชชีสามารถนิยามแบบเรียกซ้ำได้ดังนี้:

$$
\begin{aligned}
F(0) &= 0 \quad \text{(base case)} \\
F(1) &= 1 \quad \text{(base case)} \\
F(n) &= F(n-1) + F(n-2) \quad \text{สำหรับ } n > 1 \quad \text{(recursive case)}
\end{aligned}
$$

ตัวอย่าง:

```python
def fibonacci(n):
    if n == 0:
        return 0  # Base case
    elif n == 1:
        return 1  # Base case
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)  # Recursive case

#
print(fibonacci(6))  # Output: 8
```

Listing 5.25: ลำดับฟีโบนัชชีแบบเรียกซ้ำ

5.4.7 สรุป

ฟังก์ชันแบบเรียกซ้ำเป็นเทคนิคที่ทรงพลังในการแก้ปัญหาซับซ้อน โดยแยกเป็นปัญหาย่อยและแก้ซ้ำจนได้คำตอบสุดท้าย ถึงแม้จะใช้งานง่ายในบางกรณี แต่ก็ต้องระวังเรื่องประสิทธิภาพและหน่วยความจำ การเข้าใจการใช้ recursion อย่างถูกต้องจะช่วยให้สามารถเขียนโปรแกรมที่ยืดหยุ่นและแก้ปัญหาได้หลากหลายมากขึ้น

5.5 ฟังก์ชันที่ใช้บ่อยใน Python (Python Common Functions)

Python มีชุดของฟังก์ชันในตัวที่หลากหลาย ซึ่งช่วยให้การเขียนโปรแกรมง่ายขึ้นโดยไม่จำเป็นต้องสร้างฟังก์ชันเหล่านี้ใหม่ ฟังก์ชันเหล่านี้ช่วยเพิ่มความชัดเจนในการเขียนโค้ด ดูแลรักษาง่าย และมีประสิทธิภาพมากขึ้น ด้านล่างนี้คือตัวอย่างและรายการฟังก์ชันในตัวที่พบได้บ่อยใน Python

ตัวอย่างฟังก์ชันทั่วไป

1. print()

ใช้แสดงข้อความหรือข้อมูลอื่น ๆ บนหน้าจอคอนโซล

ตัวอย่าง:

```
print("Hello, World!")
```

Listing 5.26: การใช้ฟังก์ชัน print()

2. len()

คืนค่าความยาวของอ็อบเจกต์ เช่น list, string, หรือ tuple

ตัวอย่าง:

```
my_list = [1, 2, 3]
print(len(my_list))  # Output: 3
```

Listing 5.27: การใช้ฟังก์ชัน len()

3. type()

คืนค่าประเภทของอ็อบเจกต์

ตัวอย่าง:

```
print(type(10))  # Output: <class 'int'>
```

Listing 5.28: การใช้ฟังก์ชัน type()

4. sum()

คำนวณผลรวมของทุกค่าภายใน iterable

ตัวอย่าง:

```
numbers = [1, 2, 3]
print(sum(numbers))  # Output: 6
```

Listing 5.29: การใช้ฟังก์ชัน sum()

5. `input()`

รับข้อมูลจากผู้ใช้และคืนค่าเป็น string

ตัวอย่าง:

```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

Listing 5.30: การใช้ฟังก์ชัน input()

6. `max()` และ `min()`

คืนค่ามากที่สุดและน้อยที่สุดจากข้อมูลใน iterable ตามลำดับ

ตัวอย่าง:

```
numbers = [1, 2, 3]
print(max(numbers))  # Output: 3
print(min(numbers))  # Output: 1
```

Listing 5.31: การใช้ฟังก์ชัน max() และ min()

รายการฟังก์ชันในตัวของ Python (Built-in Functions)

Function	Function	Function	Function	Function
abs()	dict()	help()	min()	slice()
all()	dir()	hex()	next()	sorted()
any()	divmod()	id()	object()	staticmethod()
ascii()	enumerate()	input()	oct()	str()
bin()	eval()	int()	open()	sum()
bool()	exec()	isinstance()	ord()	super()
breakpoint()	filter()	issubclass()	pow()	tuple()
bytearray()	float()	iter()	print()	type()
bytes()	format()	len()	property()	vars()
callable()	frozenset()	list()	range()	zip()
chr()	getattr()	locals()	repr()	import()
classmethod()	globals()	map()	reversed()	
compile()	hasattr()	max()	round()	
complex()	hash()	memoryview()	set()	
delattr()	help()	min()	setattr()	
dict()	dir()	hex()	next()	

ฟังก์ชันในตัวของ Python ช่วยให้การเขียนโปรแกรมเป็นไปอย่างสะดวกและมีประสิทธิภาพมากขึ้น การเข้าใจและใช้งานฟังก์ชันเหล่านี้ได้อย่างคล่องแคล่วเป็นทักษะพื้นฐานที่สำคัญสำหรับนักพัฒนา Python ทุกคน

5.6 ตัวอย่างการใช้งานจริง

ตัวอย่างในบทนี้แสดงให้เห็นถึงการประยุกต์ใช้แนวคิดของฟังก์ชันในสถานการณ์จริง เช่น การคำนวณพื้นที่ของวงกลม การตรวจสอบจำนวนเฉพาะ และการสร้างเครื่องคิดเลขอย่างง่าย แบบฝึกหัดเหล่านี้ช่วยเสริมความเข้าใจในการนิยามฟังก์ชัน การใช้พารามิเตอร์ และการคืนค่า ตลอดจนการจัดโค้ดให้เป็นระบบแบบโมดูลาร์ในภาษา Python

5.6.1 การคำนวณพื้นที่วงกลม

การใช้ฟังก์ชันเพื่อคำนวณและคืนค่าพื้นที่ของวงกลมจากรัศมี โดยนิยามฟังก์ชันที่รับค่ารัศมีเป็นพารามิเตอร์ แล้วคำนวณพื้นที่โดยใช้สูตร πr^2 และคืนค่าพื้นที่ที่คำนวณได้ วิธีนี้ช่วยจัดเก็บตรรกะทางคณิตศาสตร์ไว้ในฟังก์ชันที่สามารถนำกลับมาใช้ซ้ำได้

ตัวอย่าง:

```
def area_of_circle(radius):
    pi = 3.14159
    return pi * (radius ** 2)

# Calling the function and printing the result
radius = 5
print(f"The area of the circle with radius {radius} is {area_of_circle(
    radius)}")
```

Listing 5.32: Function Calculating the Area of a Circle

5.6.2 ตัวอย่าง: การตรวจสอบจำนวนเฉพาะ

การตรวจสอบจำนวนเฉพาะทำได้โดยการสร้างฟังก์ชันเพื่อตรวจสอบว่าตัวเลขที่กำหนดเป็นจำนวนเฉพาะหรือไม่ โดยวนลูปตรวจสอบตัวหารตั้งแต่ 2 ถึงรากที่สองของตัวเลข หากไม่พบตัวหารแสดงว่าเป็นจำนวนเฉพาะ

ตัวอย่าง:

```
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            return False
    return True

# Testing the function
number = 29
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

Listing 5.33: Function Checking for Prime Numbers

จำนวนเฉพาะ คือ จำนวนธรรมชาติที่มากกว่า 1 ซึ่งมีตัวหารที่หารลงตัวได้เพียง 1 และตัวมันเองเท่านั้น เช่น 2, 3, 5, 7, 11 และ 13

5.6.3 ตัวอย่าง: การแยกโค้ดเป็นโมดูล

การแยกโค้ดให้เป็นโมดูล คือ การแบ่งโปรแกรมขนาดใหญ่ให้เป็นฟังก์ชันย่อยที่ทำงานเฉพาะด้าน ซึ่งสามารถพัฒนา ทดสอบ และดูแลรักษาแยกกันได้ วิธีนี้ช่วยให้โค้ดอ่านง่าย ใช้งานซ้ำได้ และดูแลรักษาง่าย เหมาะกับการพัฒนาโปรแกรมขนาดใหญ่

ตัวอย่างที่ซับซ้อน:

```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Error: Division by zero"
    return a / b

# Using the functions
num1 = 10
num2 = 5
print(f"Addition: {add(num1, num2)}")
print(f"Subtraction: {subtract(num1, num2)}")
print(f"Multiplication: {multiply(num1, num2)}")
print(f"Division: {divide(num1, num2)}")
```

Listing 5.34: Modularizing Code

ตัวอย่าง: ระบบบัญชีอย่างง่าย

```python
# Dictionary to store account details
accounts = {}

# Function to create a new account
def create_account(account_number, initial_balance=0):
    if account_number in accounts:
        return "Account already exists."
    accounts[account_number] = initial_balance
    return "Account created successfully."

# Function to deposit money
def deposit(account_number, amount):
    if account_number not in accounts:
        return "Account does not exist."
    if amount <= 0:
        return "Deposit amount must be positive."
    accounts[account_number] += amount
    return f"Deposited {amount} successfully."

# Function to withdraw money
def withdraw(account_number, amount):
    if account_number not in accounts:
        return "Account does not exist."
    if amount <= 0:
        return "Withdrawal amount must be positive."
    if accounts[account_number] < amount:
        return "Insufficient balance."
    accounts[account_number] -= amount
    return f"Withdrew {amount} successfully."

# Function to check balance
def check_balance(account_number):
    if account_number not in accounts:
        return "Account does not exist."
    return f"Balance: {accounts[account_number]}"

# Example usage
print(create_account("12345", 1000))  # Output: Account created
    successfully.
print(deposit("12345", 500))          # Output: Deposited 500 successfully.
print(withdraw("12345", 200))         # Output: Withdrew 200 successfully.
print(check_balance("12345"))         # Output: Balance: 1300
print(withdraw("12345", 2000))        # Output: Insufficient balance.
print(check_balance("67890"))         # Output: Account does not exist.
```

Listing 5.35: Complex Modularizing Code

บทที่ 5 โจทย์และแบบฝึกหัด: ฟังก์ชันและการแบ่งโปรแกรมเป็นโมดูล

5.1 การนิยามฟังก์ชัน
เขียนฟังก์ชันชื่อ `calculate_factorial` *ที่รับพารามิเตอร์* `n` *และคืนค่าแฟกทอเรียลของ* `n`

5.2 การสร้างฟังก์ชันอย่างง่าย
เขียนฟังก์ชันชื่อ `display_date` *ที่แสดงวันที่และเวลาปัจจุบันในรูปแบบ* `"YYYY-MM-DD HH:MM:SS"`

5.3 ฟังก์ชันที่มีพารามิเตอร์
เขียนฟังก์ชันชื่อ `is_even` *ที่รับค่าตัวเลขและคืนค่า* `True` *หากเป็นเลขคู่ มิฉะนั้นให้คืนค่า* `False`

5.4 ฟังก์ชันที่มีหลายพารามิเตอร์
เขียนฟังก์ชันชื่อ `calculate_gcd` *ที่รับพารามิเตอร์สองค่าและคืนค่าหรม (GCD) ของทั้งสอง*

5.5 พารามิเตอร์เริ่มต้น
เขียนฟังก์ชันชื่อ `send_email` *ที่รับพารามิเตอร์* `recipient` *และ* `subject` *(มีค่าเริ่มต้นเป็น* `"No Subject"`*) และแสดงข้อความ "Email sent to [recipient] with subject '[subject]'"*

5.6 ค่าที่ส่งกลับจากฟังก์ชัน
เขียนฟังก์ชันชื่อ `calculate_bmi` *ที่รับน้ำหนัก (กก.) และส่วนสูง (เมตร) และคืนค่า BMI*

5.7 ขอบเขตของฟังก์ชัน
เขียนฟังก์ชันชื่อ `calculate_average` *ที่สร้างตัวแปรท้องถิ่นชื่อ* `total_sum` *เพื่อใช้คำนวณค่าเฉลี่ยของรายการตัวเลขที่รับมา*

5.8 ตัวแปรสากล (Global Variables)
กำหนดตัวแปรสากลชื่อ `exchange_rate` *และเขียนฟังก์ชันชื่อ* `convert_currency` *เพื่อคำนวณอัตราแลกเปลี่ยน*

5.9 การแก้ไขตัวแปรสากล
กำหนดตัวแปรสากลชื่อ `total_users` *เริ่มต้นเป็น 0 และเขียนฟังก์ชันชื่อ* `register_user` *เพื่อเพิ่มค่าทีละ 1 และคืนค่าผลลัพธ์*

5.10 ฟังก์ชันที่มีพารามิเตอร์เริ่มต้นและหลายพารามิเตอร์
เขียนฟังก์ชันชื่อ `calculate_trip_cost` *ที่รับระยะทาง ค่าประสิทธิภาพน้ำมัน (ค่าเริ่มต้น 10 กม./ลิตร) และราคาน้ำมันต่อลิตร (ค่าเริ่มต้น 1.5) และคืนค่าค่าใช้จ่ายในการเดินทาง*

ภาคผนวก

ภาคผนวกนี้จัดเตรียมแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งานจริง เพื่อเสริมสร้างความเข้าใจเกี่ยวกับแนวคิดในบทที่ 5

A5.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Python Crash Course* โดย Eric Matthes
- *Fluent Python* โดย Luciano Ramalho

บทเรียนออนไลน์:

- Python Functions - W3Schools
- Understanding Functions in Python - Real Python

หลักสูตร:

- Coursera: Python Data Structures โดย University of Michigan
- edX: Python for Data Science โดย Microsoft

A5.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). Python Documentation - Functions. ดึงข้อมูลจาก https://docs.python.org/3/tutorial/controlflow.html#defining-functions
- Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook* (3rd ed.). O'Reilly Media.

A5.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การนิยามฟังก์ชันพื้นฐาน

- เขียนฟังก์ชันที่รับตัวเลขหนึ่งตัวและคืนค่ากำลังสองของตัวเลขนั้น
- สร้างฟังก์ชันที่รับตัวเลขสองตัวและคืนค่าผลรวมของตัวเลข

แบบฝึกหัดที่ 2: ฟังก์ชันและพารามิเตอร์

- เขียนฟังก์ชันที่รับลิสต์ของตัวเลขและคืนค่าสูงสุด
- เขียนฟังก์ชันที่รับข้อความ (string) และคืนค่าสตริงที่กลับด้าน

แบบฝึกหัดที่ 3: หลายพารามิเตอร์และหลายค่าผลลัพธ์

- เขียนฟังก์ชันที่รับตัวเลขสามตัวและคืนค่าค่าเฉลี่ย
- เขียนฟังก์ชันที่รับลิสต์ของตัวเลขและคืนค่าผลรวมและค่าเฉลี่ย

แบบฝึกหัดที่ 4: ฟังก์ชันขั้นสูง

- เขียนฟังก์ชันแบบเวียนกลับ (recursive) เพื่อหาค่าแฟกทอเรียลของตัวเลข
- สร้างฟังก์ชันที่รับลิสต์ของตัวเลขและค่าที่ต้องการค้นหา แล้วคืนค่าลิสต์ของตำแหน่งที่พบค่าดังกล่าว

A5.4 ตัวอย่างการใช้งานจริง

ตัวอย่างที่ 1: การนิยามฟังก์ชันพื้นฐาน

```python
# Function to calculate the square of a number
def square(number):
    return number ** 2

# Test the function
print(square(4))  # Output: 16
```

Listing 5.36: Function to calculate the square of a number

ตัวอย่างที่ 2: ฟังก์ชันที่ใช้พารามิเตอร์

```python
# Function to find the maximum number in a list
def find_max(numbers):
    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number

# Test the function
print(find_max([1, 2, 3, 4, 5]))  # Output: 5
```

Listing 5.37: Function to find the maximum number in a list

ตัวอย่างที่ 3: ฟังก์ชันที่ส่งคืนหลายค่า

```python
# Function to calculate sum and average of a list of numbers
def sum_and_average(numbers):
    total = sum(numbers)
    average = total / len(numbers)
    return total, average

# Test the function
numbers = [10, 20, 30, 40, 50]
total, avg = sum_and_average(numbers)
print(f"Sum: {total}, Average: {avg}")  # Output: Sum: 150, Average: 30.0
```

Listing 5.38: Function to calculate sum and average of a list of numbers

ตัวอย่างที่ 4: ฟังก์ชันเวียนกลับ

```python
# Recursive function to calculate the factorial of a number
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Test the function
print(factorial(5))  # Output: 120
```

Listing 5.39: Recursive function to calculate the factorial of a number

บทที่ 6

ลิสต์และการดำเนินการพื้นฐาน

บทนี้นำเสนอภาพรวมของลิสต์และการดำเนินการพื้นฐานในภาษา Python อย่างครอบคลุม รวมถึงการสร้างและเข้าถึงลิสต์ เมธอดทั่วไปของลิสต์ การใช้ slicing และ indexing พร้อมตัวอย่างการใช้งานจริง เพื่อให้ผู้อ่านเข้าใจวิธีจัดการและใช้งานลิสต์ได้อย่างมีประสิทธิภาพในงานเขียนโปรแกรม Python

เริ่มจากพื้นฐาน เนื้อหาอธิบายวิธีการสร้างลิสต์ โดยแสดงไวยากรณ์และวิธีการเติมข้อมูลประเภทต่าง ๆ ลงในลิสต์ ซึ่งเป็นความรู้พื้นฐานที่จำเป็นสำหรับผู้ที่ต้องการจัดการข้อมูลในรูปแบบกลุ่มในภาษา Python ส่วนของการเข้าถึงลิสต์นั้น ครอบคลุมทั้งการใช้ index แบบบวกและลบ เพื่อให้ผู้อ่านสามารถดึงข้อมูลจากตำแหน่งใด ๆ ได้อย่างมีประสิทธิภาพ

นอกจากนี้ ยังสำรวจเมธอดมาตรฐานของลิสต์ เช่น append, insert, remove, pop, clear, sort และ reverse โดยแต่ละเมธอดมีตัวอย่างชัดเจน แสดงให้เห็นการใช้งานจริง และวิธีการจัดการสมาชิกในลิสต์แบบไดนามิก

ส่วนของ slicing ให้รายละเอียดเกี่ยวกับการดึงส่วนย่อยของลิสต์ด้วยเทคนิคต่าง ๆ เช่น การใช้ค่าก้าวกระโดด (step) และ index เชิงลบ ซึ่งจำเป็นสำหรับการวิเคราะห์และจัดการข้อมูล

สุดท้าย เนื้อหายังรวมตัวอย่างการใช้งานจริง เช่น การหาค่ามากที่สุดในลิสต์ การลบค่าซ้ำ การใช้ list comprehensions การประยุกต์ในสถานการณ์จริง และการฝึกปฏิบัติ เพื่อให้ผู้อ่านเข้าใจแนวคิดเชิงทฤษฎี และสามารถประยุกต์ใช้ได้จริงในการแก้ปัญหาทางโปรแกรมมิ่ง

6.1 บทนำสู่ลิสต์

ลิสต์เป็นโครงสร้างข้อมูลที่ใช้งานบ่อยที่สุดอย่างหนึ่งในภาษา Python เหมาะสำหรับทั้งผู้เริ่มต้นและนักพัฒนาที่มีประสบการณ์ โดยลิสต์เป็นโครงสร้างข้อมูลที่มีลำดับ (ordered) กล่าวคือ สมาชิกในลิสต์จะอยู่ในลำดับที่แน่นอน และจะไม่เปลี่ยนแปลงเว้นแต่จะมีการแก้ไขอย่างเจาะจง ซึ่งคุณลักษณะนี้ช่วยให้สามารถเข้าถึงสมาชิกได้อย่างสม่ำเสมอ นอกจากนี้ลิสต์ยังสามารถเปลี่ยนแปลงได้ (mutable) หมายความว่าเนื้อหาภายในสามารถเปลี่ยนได้หลังจากการสร้าง ซึ่งมีประโยชน์อย่างมากในการอัปเดตข้อมูล การจัดเรียง หรือการสร้างโครงสร้างข้อมูลแบบไดนามิกระหว่างการทำงานของโปรแกรม

คุณสมบัติสำคัญอีกประการของลิสต์คือสามารถเก็บข้อมูลได้หลายประเภทภายในลิสต์เดียว เช่น จำนวนเต็ม ตัวเลขทศนิยม สตริง หรือแม้แต่ลิสต์อื่น ๆ ทำให้มีความยืดหยุ่นสูง ตัวอย่างเช่น ลิสต์เดียวสามารถเก็บทั้งจำนวนเต็ม สตริง และลิสต์ย่อยได้พร้อมกัน ซึ่งช่วยให้สามารถจัดระเบียบข้อมูลที่ซับซ้อนได้ เช่น การสร้างเมทริกซ์หรือต้นไม้ข้อมูล (tree)

นอกจากนี้ ลิสต์ใน Python ยังมีเมธอดในตัวที่หลากหลาย ช่วยให้การเพิ่ม ลบ หรือค้นหาข้อมูลเป็นเรื่องง่าย ส่งผลให้ลิสต์เป็นเครื่องมือสำคัญและทรงพลังในงานเขียนโปรแกรมที่หลากหลาย ตั้งแต่การเก็บข้อมูลอย่างง่ายไปจนถึงการสร้างอัลกอริธึมที่ซับซ้อน

แนวคิดสำคัญของลิสต์:

- เป็นโครงสร้างข้อมูลที่มีลำดับและเปลี่ยนแปลงได้
- สามารถเก็บข้อมูลได้หลายประเภท

ตัวอย่าง:

```
fruits = ["apple", "banana", "cherry"]
numbers = [1, 2, 3, 4, 5]
mixed = ["apple", 1, 2.5, True]
```

Listing 6.1: List Example

6.1.1 การสร้างและการเข้าถึงลิสต์

คุณสามารถสร้างลิสต์ได้โดยใช้เครื่องหมายวงเล็บเหลี่ยม `[]` และใส่ค่าที่คั่นด้วยเครื่องหมายจุลภาคเข้าไป ซึ่งจะช่วยให้สามารถจัดเก็บหลายค่าภายในตัวแปรเดียว โดยลิสต์สามารถเก็บข้อมูลได้หลากหลายประเภท เช่น ตัวเลข สตริง หรือลิสต์อื่น ๆ จึงเป็นโครงสร้างข้อมูลที่ยืดหยุ่นและหลากหลาย

แนวคิดสำคัญของการสร้างและเข้าถึงลิสต์:

- ใช้วงเล็บเหลี่ยม `[]`
- ตัวอย่าง: `fruits = ["apple", "banana", "cherry"]`
- การเข้าถึงด้วยดัชนี (ทั้งบวกและลบ)
- ตัวอย่าง: `fruits[0]` (สมาชิกตัวแรก), `fruits[-1]` (สมาชิกตัวสุดท้าย)

ตัวอย่าง:

```
# Creating and accessing lists
prime_numbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
print(f"Prime numbers: {prime_numbers}")
fifth_prime = prime_numbers[4]
print(f"Fifth prime number: {fifth_prime}")
```

Listing 6.2: Creating and Accessing Lists

6.1.2 การเข้าถึงด้วยดัชนีลบ

การเข้าถึงด้วยดัชนีลบช่วยให้สามารถเข้าถึงสมาชิกจากด้านท้ายของลิสต์ได้ โดยใช้ตัวเลขติดลบ เช่น ดัชนี `-1` หมายถึงสมาชิกตัวสุดท้าย, `-2` หมายถึงตัวรองสุดท้าย เป็นต้น ซึ่งเป็นวิธีที่สะดวกและเป็นธรรมชาติในการเข้าถึงลิสต์จากด้านหลัง

ตัวอย่าง:

```
colors = ["red", "blue", "green", "yellow", "purple"]
second_to_last_color = colors[-2]
print(f"Second to last color: {second_to_last_color}")
```

Listing 6.3: Example: Negative Indexing

6.1.3 การแก้ไขลิสต์

ลิสต์สามารถเปลี่ยนแปลงได้ (mutable) หมายความว่าสามารถแก้ไขเนื้อหาภายในได้หลังจากการสร้าง ซึ่งช่วยให้สามารถปรับเปลี่ยน เพิ่ม หรือลบสมาชิกในลิสต์ตามต้องการ ทำให้ลิสต์เป็นเครื่องมือที่ยืดหยุ่นสูงสำหรับการทำงานที่หลากหลาย เช่น อัปเดตค่าทีละตัว ขยายรายการ หรือจัดเรียงสมาชิกใหม่

แนวคิดสำคัญของการสร้างและเข้าถึงลิสต์:

- การเปลี่ยนค่าด้วยดัชนี
- ตัวอย่าง: `fruits[1] = "blueberry"`

ตัวอย่าง:

```
# Modifying lists
shapes = ["circle", "square", "triangle", "rectangle", "hexagon"]
shapes[1] = "ellipse"
shapes[3] = "pentagon"
print(f"Modified shapes: {shapes}")
```

Listing 6.4: Example:Modifying lists

6.2 เมธอดของลิสต์

ลิสต์ใน Python มาพร้อมกับเมธอดในตัวหลายแบบที่ช่วยให้คุณสามารถจัดการกับลิสต์ในรูปแบบต่าง ๆ ได้ เพิ่มขีดความสามารถในการใช้งานและทำให้ลิสต์เป็นเครื่องมือที่ทรงพลังในการจัดการข้อมูล ตัวอย่างเช่น เมธอด `append()` และ `extend()` ใช้สำหรับเพิ่มสมาชิกใหม่ไปยังท้ายลิสต์ ส่วน `insert()` ช่วยเพิ่มสมาชิกในตำแหน่งที่ระบุ นอกจากนี้คุณยังสามารถลบสมาชิกได้ด้วยเมธอด เช่น `remove()` ที่ลบค่าที่พบครั้งแรกตามที่ระบุ และ `pop()` ที่ลบและคืนค่าจากตำแหน่งที่ระบุหรือจากท้ายลิสต์หากไม่ระบุดัชนี เมธอดเหล่านี้ช่วยให้สามารถจัดการกับเนื้อหาของลิสต์ได้อย่างยืดหยุ่นระหว่างการทำงานของโปรแกรม

ลิสต์ยังมีเมธอดสำหรับค้นหาและจัดเรียงข้อมูล ทำให้สามารถจัดระเบียบและเข้าถึงข้อมูลได้ง่ายขึ้น เมธอด `index()` ใช้ค้นหาตำแหน่งของค่าที่ระบุครั้งแรก และ `count()` ใช้นับจำนวนครั้งที่ค่าปรากฏในลิสต์ สำหรับการจัดเรียง เมธอด `sort()` จะจัดเรียงค่าจากน้อยไปมากโดยปริยาย แต่สามารถปรับให้เรียงจากมากไปน้อยหรือกำหนดเกณฑ์การเรียงแบบอื่นได้ เมธอด `reverse()` ใช้ในการสลับลำดับของสมาชิกในลิสต์ ซึ่งมีประโยชน์ในหลายสถานการณ์ เมธอดทั้งหมดนี้ทำให้ลิสต์เป็นโครงสร้างข้อมูลที่ยืดหยุ่นและจำเป็นในภาษา Python ที่สามารถจัดการข้อมูลได้อย่างมีประสิทธิภาพ

เมธอดทั่วไปของลิสต์:

- `append()`: เพิ่มค่าหนึ่งค่าไปที่ท้ายลิสต์
- `extend()`: เพิ่มสมาชิกทั้งหมดจากลิสต์อื่นไปที่ท้ายลิสต์ปัจจุบัน
- `insert()`: แทรกค่าที่ตำแหน่งที่ระบุ
- `remove()`: ลบค่าที่พบครั้งแรกของค่าที่ระบุ
- `pop()`: ลบและคืนค่าจากตำแหน่งที่ระบุ
- `index()`: คืนค่าตำแหน่งแรกที่พบค่าที่ระบุ
- `count()`: นับจำนวนครั้งที่ค่าปรากฏในลิสต์
- `clear()`: ลบค่าทั้งหมดในลิสต์
- `sort()`: จัดเรียงค่าจากน้อยไปมาก
- `reverse()`: สลับลำดับของสมาชิกในลิสต์

6.2.1 Append

เมธอด `append()` ใช้เพิ่มค่าหนึ่งค่าไปที่ท้ายของลิสต์ ซึ่งช่วยให้สามารถเพิ่มสมาชิกใหม่ได้แบบไดนามิก เหมาะสำหรับการสร้างลิสต์ทีละค่าภายในลูป หรือจากอินพุตของผู้ใช้ โดยรับประกันว่าสมาชิกใหม่จะถูกเพิ่มต่อท้ายเสมอ

ตัวอย่าง:

```
# Append method
fruits = ["apple", "banana", "cherry"]
more_fruits = ["mango", "pineapple"]
for fruit in more_fruits:
    fruits.append(fruit)
print(f"Fruits after append: {fruits}")
# Output: Fruits after append: ['apple', 'banana', 'cherry', 'mango', '
    pineapple']
```

Listing 6.5: Example: List Append

6.2.2 Insert

เมธอด `insert()` ใช้เพิ่มสมาชิกใหม่ลงในตำแหน่งที่ระบุภายในลิสต์ ให้คุณสามารถควบคุมตำแหน่งการเพิ่มค่าได้อย่างแม่นยำ โดยสมาชิกที่อยู่เดิมจะถูกเลื่อนไปทางขวาเพื่อเปิดที่ว่างสำหรับสมาชิกใหม่

ตัวอย่าง:

```
# Insert method
berries = ["raspberry", "blackberry"]
berries.insert(1, "strawberry")
berries.insert(2, "blueberry")
print(f"Berries after insert: {berries}")
# Output: Berries after insert: ['raspberry', 'strawberry', 'blueberry', '
    blackberry']
```

Listing 6.6: Example: List Insert

6.2.3 Remove

เมธอด `remove()` ใช้ลบค่าที่พบครั้งแรกของค่าที่ระบุจากลิสต์ โดยไม่ต้องทราบตำแหน่งที่แน่นอน ถ้าค่าที่ระบุปรากฏหลายครั้ง จะลบแค่ครั้งแรกเท่านั้น และสมาชิกถัดไปจะถูกเลื่อนเข้ามาแทนที่

ตัวอย่าง:

```
# Remove method
fruits_with_duplicates = ["apple", "banana", "apple", "cherry", "apple", "
    kiwi"]
while "apple" in fruits_with_duplicates:
    fruits_with_duplicates.remove("apple")
print(f"Fruits after remove: {fruits_with_duplicates}")
# Output: Fruits after remove: ['banana', 'cherry', 'kiwi']
```

Listing 6.7: Example: List Remove

6.2.4 Pop

เมธอด `pop()` ใช้ลบและคืนค่าของสมาชิกที่ตำแหน่งที่ระบุจากลิสต์ ทำให้สามารถลบและใช้งานค่าดังกล่าวได้ในคำสั่งเดียว หากไม่ระบุตำแหน่ง จะลบและคืนค่าสมาชิกสุดท้ายในลิสต์ วิธีนี้มีประโยชน์โดยเฉพาะในกรณีที่ทำงานแบบสแต็ก (stack)

ตัวอย่าง:

```
# Pop method
grades = [85, 90, 78, 92, 88]
third_grade = grades.pop(2)
grades.append(third_grade)
print(f"Grades after pop: {grades}")
# Output: Grades after pop: [85, 90, 92, 88, 78]
```

Listing 6.8: Example: List Pop

6.2.5 Index

เมธอด `index()` ใช้ค้นหาตำแหน่งแรกที่พบค่าที่ระบุในลิสต์ หากไม่พบค่าจะเกิด `ValueError` สามารถระบุตำแหน่งเริ่มต้นเพื่อค้นหาค่าซ้ำถัดไปในลิสต์ได้ เหมาะสำหรับค้นหาข้อมูลเฉพาะภายในลิสต์อย่างมีประสิทธิภาพ

ตัวอย่าง:

```
1
2  # Using index() to find the first occurrence of "dog"
3  animals = ["cat", "dog", "rabbit", "hamster", "dog", "parrot"]
4  first_dog_index = animals.index("dog")
5  print(f"The first occurrence of 'dog' is at index: {first_dog_index}")
6  # Output: The first occurrence of 'dog' is at index: 1
7
8  # Using index() to find the second occurrence of "dog"
9  second_dog_index = animals.index("dog", first_dog_index + 1)
10 print(f"The second occurrence of 'dog' is at index: {second_dog_index}")
11 # Output: The second occurrence of 'dog' is at index: 4
```

Listing 6.9: Example: List Index

6.2.6 Clear

เมธอด `clear()` ใช้ลบสมาชิกทั้งหมดในลิสต์ ทำให้ลิสต์ว่างเปล่าโดยไม่ต้องสร้างลิสต์ใหม่ เหมาะสำหรับการรีเซตค่าภายในลิสต์โดยยังคงใช้โครงสร้างเดิม

ตัวอย่าง:

```
1  # Clear method
2  nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3  for sublist in nested_list:
4      sublist.clear()
5  print(f"Nested list after clear: {nested_list}")
6  # Output: Nested list after clear: [[], [], []]
```

Listing 6.10: Example: List Clear

6.2.7 Sort

เมธอด `sort()` ใช้จัดเรียงสมาชิกในลิสต์จากน้อยไปหามากโดยปริยาย สามารถปรับให้เรียงจากมากไปหาน้อยหรือกำหนดเกณฑ์การจัดเรียงอื่น ๆ ได้ การเรียงลิสต์ช่วยให้การจัดการ เปรียบเทียบ และวิเคราะห์ข้อมูลมีประสิทธิภาพมากขึ้น

ตัวอย่าง:

```
1  numbers = [4, 2, 3, 1, 5]
2  numbers.sort()
3  print(numbers)  # Output: [1, 2, 3, 4, 5]
```

Listing 6.11: Example: List Sort

6.2.8 Reverse

เมธอด `reverse()` ใช้สลับลำดับของสมาชิกในลิสต์ โดยสมาชิกตัวแรกจะกลายเป็นตัวสุดท้าย และตัวสุดท้ายจะกลายเป็นตัวแรก เมธอดนี้จะเปลี่ยนแปลงลิสต์เดิมทันที เหมาะสำหรับการประมวลผลข้อมูลในลำดับย้อนกลับ

ตัวอย่าง:

```
numbers = [1, 2, 3, 4, 5]
numbers.reverse()
print(numbers)  # Output: [5, 4, 3, 2, 1]
```

Listing 6.12: Example: List Reverse

6.3 การตัดช่วงข้อมูล (Slicing)

การเข้าถึงบางส่วนของลิสต์ในภาษา Python สามารถทำได้โดยใช้ไวยากรณ์การตัดช่วง `string[start: stop: step]` ซึ่งช่วยให้คุณสามารถดึงข้อมูลช่วงหนึ่งจากลิสต์หรือสตริงตามดัชนีและค่าก้าวที่กำหนด มาดูรายละเอียดของแต่ละพารามิเตอร์ :

คำอธิบาย:

- **start**: ดัชนีเริ่มต้นของช่วงที่ต้องการตัด หากไม่ระบุ จะเริ่มจากตำแหน่งแรกของลิสต์
- **stop**: ดัชนีสิ้นสุดของช่วง (ไม่รวมตำแหน่งที่ระบุ) หากไม่ระบุ จะสิ้นสุดที่ตำแหน่งสุดท้ายของลิสต์
- **stride**: ค่าก้าว (จำนวนตำแหน่งที่จะข้ามไปในแต่ละขั้น) หากไม่ระบุ ค่าปริยายคือ 1 ซึ่งหมายถึงดึงค่าทุกตำแหน่งที่ต่อกัน

ตัวอย่าง:

```
data = list(range(100))
sliced_data = data[10:51:5]
print(f"Sliced data: {sliced_data}")
```

Listing 6.13: Example: List Slicing

6.3.1 การเข้าถึงบางส่วนของลิสต์

ไวยากรณ์:

```
list[start:stop:step]
```

ตัวอย่าง:

```
# Slicing a list
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Slicing from index 2 to 5
print(numbers[2:6])  # Output: [2, 3, 4, 5]

# Slicing with step
print(numbers[1:8:2])  # Output: [1, 3, 5, 7]

# Slicing from start to a position
print(numbers[:4])  # Output: [0, 1, 2, 3]

# Slicing from a position to the end
print(numbers[6:])  # Output: [6, 7, 8, 9]
```

Listing 6.14: Example: List Slicing

6.3.2 การตัดช่วงด้วยดัชนีลบ

คุณสามารถใช้ดัชนีลบเพื่อตัดช่วงข้อมูลจากด้านท้ายของลิสต์ ซึ่งช่วยให้สามารถเข้าถึงสมาชิกจากท้ายรายการได้อย่างสะดวก ตัวอย่างเช่น `list[-3:]` จะดึงสมาชิกสามตัวสุดท้ายของลิสต์ วิธีนี้ช่วยให้สามารถจัดการกับลิสต์ได้อย่างยืดหยุ่นและเป็นธรรมชาติ

ตัวอย่าง:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Negative slicing
print(numbers[-5:-1])  # Output: [5, 6, 7, 8]

# Slicing with negative step
print(numbers[::-1])  # Output: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Listing 6.15: Example: Negative Slicing

6.3.3 การเข้าถึงบางส่วนของสตริง

พิจารณาสตริง `"Frozen Watermelon"` มาดูตัวอย่างการใช้ไวยากรณ์ `start:stop:step` ในการตัดช่วงข้อความ:

ตัวอย่าง: `String[0:6]`

```
string = "Frozen Watermelon"
substring = string[0:6]
print(substring)  # Output: "Frozen"
```

Listing 6.16: Extracting a substring from index 0 to 6

คำอธิบาย: เริ่มที่ตำแหน่ง 0 และหยุดก่อนตำแหน่งที่ 6 ได้ข้อความ `"Frozen"`

ตัวอย่าง: `String[7:]`

```
string = "Frozen Watermelon"
substring = string[7:]
print(substring)  # Output: "Watermelon"
```

Listing 6.17: Extracting a substring from index 7 to the end

คำอธิบาย: เริ่มที่ตำแหน่งที่ 7 จนถึงท้ายข้อความ ได้ `"Watermelon"`

ตัวอย่าง: `String[::2]`

```
string = "Frozen Watermelon"
substring = string[::2]
print(substring)  # Output: "Foe aemln"
```

Listing 6.18: Extracting every second character from the entire string

คำอธิบาย: ดึงทุกตัวที่สองโดยเริ่มจากตัวแรก โดยข้ามหนึ่งตัวอักษรระหว่างกัน

ตัวอย่าง: `String[0:12:3]`

```
string = "Frozen Watermelon"
substring = string[0:12:3]
print(substring)  # Output: "Fz t"
```

Listing 6.19: Extracting a substring from index 0 to 12 with a step of 3

คำอธิบาย: เริ่มที่ตำแหน่ง 0 และหยุดก่อนตำแหน่งที่ 12 โดยใช้ก้าว 3 ดึงตัวอักษรทุกสามตัว ได้ข้อความ `"Fz t"`

ตัวอย่าง: `String[::-1]`

```
string = "Frozen Watermelon"
substring = string[::-1]
print(substring)  # Output: "nolemretaW nezorF"
```

Listing 6.20: Reversing the entire string

คำอธิบาย: พลิกกลับลำดับของข้อความโดยเริ่มจากตัวท้ายไปต้น ใช้ก้าว -1 เพื่อย้อนกลับข้อความทั้งหมด

6.4 ฟังก์ชันในตัวสำหรับการทำงานกับลิสต์

ในภาษา Python ฟังก์ชันในตัว (built-in functions) มีบทบาทสำคัญในการทำให้การเขียนโปรแกรมเป็นเรื่องง่าย โดยเฉพาะอย่างยิ่งเมื่อต้องทำงานกับลิสต์ ฟังก์ชันเหล่านี้ช่วยให้สามารถดำเนินการต่าง ๆ ได้อย่างมีประสิทธิภาพ ตั้งแต่การคำนวณพื้นฐาน เช่น ความยาว ผลรวม ค่าสูงสุด และค่าต่ำสุด ไปจนถึงการจัดเรียง ย้อนลำดับ และวนซ้ำ พร้อมดัชนี ด้วยการใช้ฟังก์ชันในตัวเหล่านี้ นักพัฒนาสามารถเขียนโค้ดที่กระชับ อ่านง่าย และดูแลรักษาได้สะดวก จึงถือเป็นเครื่องมือสำคัญในงานเขียนโปรแกรมด้วย Python

โปรแกรมตัวอย่างนี้แสดงการใช้ฟังก์ชันในตัวที่ใช้บ่อยสำหรับการดำเนินการต่าง ๆ กับลิสต์ในภาษา Python

ตัวอย่าง:

```python
# Example list
numbers = [4, 2, 9, 1, 5, 6]

# 1. len(): Get the length of the list
length = len(numbers)
print(f"Length of the list: {length}")  # Output: Length of the list: 6

# 2. sum(): Calculate the sum of all elements in the list
total_sum = sum(numbers)
print(f"Sum of all elements: {total_sum}")  # Output: Sum of all elements:
    27

# 3. max(): Find the maximum value in the list
max_value = max(numbers)
print(f"Maximum value: {max_value}")  # Output: Maximum value: 9

# 4. min(): Find the minimum value in the list
min_value = min(numbers)
print(f"Minimum value: {min_value}")  # Output: Minimum value: 1

# 5. sorted(): Return a sorted version of the list
sorted_numbers = sorted(numbers)
print(f"Sorted list: {sorted_numbers}")  # Output: Sorted list: [1, 2, 4,
    5, 6, 9]

# 6. any(): Check if any element in the list is True
bool_list = [False, True, False]
any_true = any(bool_list)
print(f"Is any element True? {any_true}")  # Output: Is any element True?
    True

# 7. all(): Check if all elements in the list are True
all_true = all(bool_list)
print(f"Are all elements True? {all_true}")  # Output: Are all elements
    True? False

# 8. list(): Convert an iterable to a list (if not already a list)
string = "hello"
char_list = list(string)
print(f"List of characters: {char_list}")  # Output: List of characters: ['
    h', 'e', 'l', 'l', 'o']

# 9. reversed(): Return a reverse iterator of the list
reversed_numbers = list(reversed(numbers))
print(f"Reversed list: {reversed_numbers}")  # Output: Reversed list: [6,
    5, 1, 9, 2, 4]

# 10. enumerate(): Return an iterator of tuples containing index and value
enumerated_numbers = list(enumerate(numbers))
print(f"Enumerated list: {enumerated_numbers}")
# Output: Enumerated list: [(0, 4), (1, 2), (2, 9), (3, 1), (4, 5), (5, 6)]
```

Listing 6.21: Using Built-in Functions on a List

สรุปฟังก์ชันในตัวที่ใช้กับลิสต์

- `len()`: คืนค่าจำนวนสมาชิกในลิสต์
- `sum()`: คืนค่าผลรวมของสมาชิกทั้งหมดในลิสต์
- `max()`: คืนค่าสูงสุดของลิสต์
- `min()`: คืนค่าต่ำสุดของลิสต์
- `sorted()`: คืนลิสต์ที่ถูกจัดเรียงจากลิสต์เดิม
- `any()`: คืนค่า `True` หากมีสมาชิกใด ๆ ในลิสต์ที่เป็น `True`
- `all()`: คืนค่า `True` ก็ต่อเมื่อสมาชิกทั้งหมดในลิสต์เป็น `True`
- `list()`: แปลงข้อมูลที่วนซ้ำได้ (เช่น สตริง) ให้เป็นลิสต์
- `reversed()`: คืนตัววนซ้ำแบบย้อนลำดับจากลิสต์
- `enumerate()`: คืนตัววนซ้ำที่ประกอบด้วยทูเพิล (ดัชนี, ค่า) จากลิสต์

6.5 ลิสต์สองมิติ

ลิสต์สองมิติ หรือที่เรียกกันว่า 2D lists หรือเมทริกซ์ (matrices) คือการสร้างลิสต์ซ้อนอยู่ภายในลิสต์อีกทีหนึ่ง ซึ่งใช้แทนโครงสร้างแบบตารางที่มีแถวและคอลัมน์ เหมือนกับข้อมูลในตารางหรือสเปรดชีต ในภาษา Python ลิสต์สองมิติคือ ลิสต์ของลิสต์ โดยที่แต่ละลิสต์ย่อยจะแทนหนึ่งแถวของเมทริกซ์

การสร้างลิสต์สองมิติ

เราสามารถสร้างลิสต์สองมิติได้โดยการซ้อนลิสต์หลายลิสต์ไว้ในลิสต์หลัก ตัวอย่างเช่น:

ตัวอย่าง:

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

Listing 6.22: Creating a Two-Dimensional List

ในตัวอย่างนี้ `matrix` เป็นลิสต์สองมิติขนาด 3x3 โดยมีข้อมูลดังนี้:

- แถวที่หนึ่ง คือ `[1, 2, 3]`
- แถวที่สอง คือ `[4, 5, 6]`
- แถวที่สาม คือ `[7, 8, 9]`

การเข้าถึงสมาชิก

การเข้าถึงสมาชิกในลิสต์สองมิติสามารถทำได้โดยใช้ดัชนีซ้อนสองระดับ ดัชนีแรกใช้ระบุตำแหน่งแถว ส่วนดัชนีที่สองใช้ระบุตำแหน่งคอลัมน์

ตัวอย่าง:

```
element = matrix[1][2]
print(element)  # Output: 6
```

Listing 6.23: Accessing Elements in a 2D List

ในที่นี้ `matrix[1][2]` หมายถึงสมาชิกในแถวที่สองและคอลัมน์ที่สาม ซึ่งก็คือค่า 6

การแก้ไขสมาชิก

การแก้ไขสมาชิกในลิสต์สองมิติทำได้เช่นเดียวกับการเข้าถึงสมาชิก:

ตัวอย่าง:

```
matrix[0][1] = 10
print(matrix)
# Output:
# [
#     [1, 10, 3],
#     [4, 5, 6],
#     [7, 8, 9]
# ]
```

Listing 6.24: Modifying Elements in a 2D List

จากตัวอย่างนี้ เราได้เปลี่ยนค่าจาก 2 เป็น 10 ที่ตำแหน่งแถวแรก คอลัมน์ที่สอง

การวนลูปผ่านลิสต์สองมิติ

เราสามารถวนลูปผ่านลิสต์สองมิติได้โดยใช้ลูปซ้อนกัน โดยลูปด้านนอกจะวนผ่านแต่ละแถว และลูปด้านในจะวนผ่านสมาชิกแต่ละตัวในแถวนั้น

ตัวอย่าง:

```
for row in matrix:
    for element in row:
        print(element, end=" ")
    print()
```

Listing 6.25: Iterating Through a Two-Dimensional List

การประยุกต์ใช้งาน

ลิสต์สองมิติถูกใช้งานอย่างแพร่หลายในหลายบริบท เช่น:

- **เมทริกซ์:** สำหรับการคำนวณทางคณิตศาสตร์ เช่น การบวกเมทริกซ์ การคูณเมทริกซ์ เป็นต้น
- **กริด (Grid):** สำหรับใช้ในเกมกระดาน เช่น กระดานหมากรุก หรือเกม XO (tic-tac-toe)
- **ตารางข้อมูล:** สำหรับจัดเก็บข้อมูลในรูปแบบแถวและคอลัมน์ เช่น ฐานข้อมูลแบบตาราง
- **ภาพ:** สำหรับเก็บค่าพิกเซลของภาพในการประมวลผลภาพ ซึ่งแต่ละสมาชิกแทนค่าพิกเซล

6.6 ทูเพิลในภาษา Python

ทูเพิล () คือโครงสร้างข้อมูลที่มีลำดับและ **ไม่สามารถเปลี่ยนแปลงได้** (immutable) ในภาษา Python หมายความว่า เมื่อสร้างทูเพิลขึ้นมาแล้ว จะไม่สามารถแก้ไข เพิ่ม หรือลบสมาชิกภายในได้ ทูเพิลคล้ายกับลิสต์ตรงที่สามารถเก็บหลายค่าภายในตัวมันได้ แต่ต่างกันตรงที่ทูเพิลไม่สามารถเปลี่ยนแปลงได้ ซึ่งความไม่เปลี่ยนแปลงนี้ทำให้ทูเพิลเป็นทางเลือกที่น่าเชื่อถือในการจัดกลุ่มข้อมูลที่ควรคงที่ตลอดการทำงานของโปรแกรม เช่น การเก็บค่าพิกัด ชุดข้อมูลที่ไม่ควรเปลี่ยนแปลง หรือระเบียนข้อมูลถาวร

แนวคิดสำคัญของ Tuple:

- **โครงสร้างที่มีลำดับ**: ทูเพิลจะคงลำดับของสมาชิกไว้
- ใช้วงเล็บ `()`
- **ไม่เปลี่ยนแปลง**: ไม่สามารถเปลี่ยนค่าภายในทูเพิลหลังจากที่สร้างแล้ว
- **ข้อมูลหลากหลายประเภท**: ทูเพิลสามารถเก็บค่าหลายประเภทผสมกันได้
- **แฮชได้ (Hashable)**: ทูเพิลสามารถใช้เป็นคีย์ในดิกชันนารีได้
- **การแพ็กและแยกแพ็กทูเพิล**: สามารถจัดกลุ่มหลายค่าลงในทูเพิล และแยกออกมาเก็บในตัวแปรแต่ละตัวได้

6.6.1 การสร้างและเข้าถึงทูเพิล

เราสามารถสร้างทูเพิลได้โดยวางค่าที่คั่นด้วยเครื่องหมายจุลภาค (comma) ไว้ในวงเล็บ `()` เช่น `my_tuple = (1, 2, 3)` จะสร้างทูเพิลที่มี 3 ค่า

การสร้างทูเพิล

คุณสามารถสร้างทูเพิลได้โดยใส่ค่าภายในวงเล็บ `()` และคั่นแต่ละค่าด้วยเครื่องหมายจุลภาค

```
# Creating a tuple
my_tuple = (1, 2, 3, "apple", "banana")
print(my_tuple)
```

Listing 6.26: Creating a tuple

การเข้าถึงสมาชิก

คุณสามารถเข้าถึงสมาชิกในทูเพิลได้โดยใช้ดัชนี เช่นเดียวกับลิสต์

```
# Accessing elements
print(my_tuple[0])  # Output: 1
print(my_tuple[3])  # Output: "apple"
```

Listing 6.27: Accessing elements in a tuple

ทูเพิลไม่สามารถเปลี่ยนแปลงได้

เมื่อสร้างทูเพิลแล้ว จะไม่สามารถเปลี่ยนแปลงสมาชิกภายในได้ เช่น การเพิ่ม ลบ หรือแก้ไขค่า

```
# Attempting to modify a tuple will result in an error
my_tuple[1] = 10  # This will raise a TypeError
```

Listing 6.28: Attempting to modify a tuple

ทูเพิลที่มีสมาชิกเพียงหนึ่งค่า

หากต้องการสร้างทูเพิลที่มีเพียงหนึ่งค่า จำเป็นต้องใส่เครื่องหมายจุลภาคหลังค่านั้นด้วย

```
# Single element tuple
single_element_tuple = (5,)
print(type(single_element_tuple))  # Output: <class 'tuple'>
```

Listing 6.29: Single element tuple

6.6.2 การแพ็กและแยกแพ็กทูเพิล

การแพ็กและแยกแพ็กทูเพิล

การแพ็กทูเพิล (Tuple Packing) คือการกำหนดค่าหลายค่าลงในตัวแปรเดียวโดยใช้เครื่องหมายจุลภาค ซึ่ง Python จะสร้างทูเพิลให้อัตโนมัติ ส่วน **การแยกแพ็กทูเพิล (Tuple Unpacking)** คือการนำค่าจากทูเพิลแยกเก็บในตัวแปรหลายตัว

```
# Tuple Packing
my_tuple = 1, 2, 3  # Parentheses are optional
print(my_tuple)  # Output: (1, 2, 3)

# Tuple Unpacking
a, b, c = my_tuple
print(a)  # Output: 1
print(b)  # Output: 2
print(c)  # Output: 3
```

Listing 6.30: Tuple packing and unpacking

ตัวอย่าง: ใช้ทูเพิลเป็นค่าที่ส่งกลับจากฟังก์ชัน

ทูเพิลมักถูกใช้ในการส่งค่าหลายค่ากลับจากฟังก์ชัน

```
# Example function returning multiple values as a tuple
def get_student_info():
    name = "Alice"
    age = 21
    grade = "A"
    return name, age, grade

# Unpacking the returned tuple
student_name, student_age, student_grade = get_student_info()
print(student_name)  # Output: Alice
print(student_age)   # Output: 21
print(student_grade) # Output: A
```

Listing 6.31: Tuple as a function return value

6.6.3 เมธอดที่ใช้บ่อยของทูเพิล

เมธอดที่ใช้บ่อยของทูเพิล ได้แก่ `count()` ซึ่งใช้เพื่อนับจำนวนครั้งที่ค่าหนึ่งปรากฏอยู่ในทูเพิล และ `index()` ซึ่งคืนตำแหน่งแรกของค่าที่กำหนด เมธอดเหล่านี้ช่วยให้สามารถวิเคราะห์และจัดการข้อมูลภายในทูเพิลได้อย่างมีประสิทธิภาพ

ตัวอย่าง:

```
numbers = (1, 2, 3, 2, 4, 2, 5)

# Counting occurrences of 2
print(numbers.count(2))  # Output: 3

# Finding the index of 3
print(numbers.index(3))  # Output: 2
```

Listing 6.32: Common Tuple Methods

เหตุผลที่ควรใช้ทูเพิล

- **ไม่สามารถเปลี่ยนแปลงได้ (Immutability)**: ทูเพิลเหมาะสำหรับใช้เมื่อคุณต้องการให้ข้อมูลคงที่ตลอดโปรแกรม
- **ประสิทธิภาพ (Efficiency)**: ทูเพิลมักทำงานได้เร็วกว่าเมื่อเปรียบเทียบกับลิสต์ เนื่องจากไม่มีการเปลี่ยนแปลงค่า
- **สามารถแฮชได้ (Hash-ability)**: เนื่องจากทูเพิลไม่สามารถเปลี่ยนแปลงได้ จึงสามารถใช้เป็นคีย์ในดิกชันนารีได้ ขณะที่ลิสต์ไม่สามารถทำได้

ตัวอย่าง: การใช้ทูเพิลเป็นคีย์ในดิกชันนารี

เนื่องจากทูเพิลไม่สามารถเปลี่ยนแปลงได้ จึงสามารถใช้เป็นคีย์ในดิกชันนารีได้ ขณะที่ลิสต์ไม่สามารถใช้เป็นคีย์ได้ เพราะคีย์ในดิกชันนารีต้องเป็นชนิดข้อมูลที่สามารถแฮชได้ (hashable) ตัวอย่างต่อไปนี้แสดงการใช้ทูเพิลเป็นคีย์เพื่อแทนพิกัดในระบบพิกัดสองมิติ:

ตัวอย่าง:

```python
# Example: Using tuples as keys in a dictionary to store points in a 2D coordinate system

# Create a dictionary where keys are (x, y) coordinates (tuples) and values are the names of points
points = {
    (0, 0): "Origin",
    (1, 2): "Point A",
    (3, 4): "Point B",
    (-1, -1): "Point C"
}

# Accessing a point by its coordinates
coordinate = (1, 2)
point_name = points[coordinate]
print(f"The point at {coordinate} is named '{point_name}'")  # Output: The point at (1, 2) is named 'Point A'

# Adding a new point
points[(5, 5)] = "Point D"

# Printing all points
for coord, name in points.items():
    print(f"Coordinate {coord} is '{name}'")

# Output:
# Coordinate (0, 0) is 'Origin'
# Coordinate (1, 2) is 'Point A'
# Coordinate (3, 4) is 'Point B'
# Coordinate (-1, -1) is 'Point C'
# Coordinate (5, 5) is 'Point D'
```

Listing 6.33: Example: Using tuples as keys in a dictionary to store points in a 2D coordinate system

คำอธิบาย:

ทูเพิลเป็นคีย์ในดิกชันนารี: ในตัวอย่างนี้ ทูเพิล `(x, y)` ถูกใช้เป็นคีย์ในดิกชันนารี `points` ซึ่งแต่ละคีย์แทนพิกัดตำแหน่งบนระนาบสองมิติ และค่าที่เกี่ยวข้องคือชื่อของจุดนั้น

ความไม่เปลี่ยนแปลง (Immutability): ทูเพิลไม่สามารถเปลี่ยนแปลงค่าได้หลังจากสร้างแล้ว คุณสมบัตินี้ทำให้ทูเพิลเหมาะสำหรับใช้เป็นคีย์ที่ต้องการความคงที่ตลอดโปรแกรม

การค้นหาที่มีประสิทธิภาพ: การใช้ทูเพิลเป็นคีย์ช่วยให้สามารถค้นหาข้อมูลโดยใช้หลายเงื่อนไขร่วมกัน เช่น พิกัด x และ y ในตัวอย่างนี้

คุณสมบัตินี้เป็นเอกลักษณ์เฉพาะของทูเพิล เพราะลิสต์ไม่สามารถใช้เป็นคีย์ได้ การที่ทูเพิลมีความไม่เปลี่ยนแปลงทำให้สามารถนำไปใช้กับโครงสร้างข้อมูลที่ต้องการความเสถียรและรองรับการแฮชได้

สรุป

ทูเพิลเป็นโครงสร้างข้อมูลที่มีประสิทธิภาพและหลากหลายในภาษา Python โดยเฉพาะอย่างยิ่งเมื่อต้องการเก็บข้อมูลที่มีลำดับและไม่ต้องการให้มีการเปลี่ยนแปลง ความไม่เปลี่ยนแปลงของทูเพิลช่วยรักษาความสมบูรณ์ของข้อมูล ทำให้

เหมาะสำหรับกรณีที่ต้องการความแม่นยำและคงที่ของข้อมูลในโปรแกรม

6.7 ตัวอย่างการใช้งานจริง

ตัวอย่างการใช้งานจริงในบทนี้จะแสดงการนำการดำเนินการกับลิสต์มาใช้ในสถานการณ์จริง เช่น การหาค่ามากที่สุดในลิสต์ซึ่งใช้การวนลูปและเปรียบเทียบค่า การลบค่าซ้ำซึ่งแสดงการใช้ลูปและเงื่อนไข และการใช้ list comprehension เพื่อสร้างลิสต์ใหม่อย่างกระชับ ตัวอย่างเหล่านี้ช่วยเสริมความเข้าใจเกี่ยวกับเมธอดของลิสต์ และแสดงให้เห็นถึงประโยชน์ในการแก้ปัญหาโปรแกรมจริงได้อย่างมีประสิทธิภาพ

6.7.1 ตัวอย่าง: การหาค่ามากที่สุดในลิสต์

การหาค่ามากที่สุดในลิสต์ต้องใช้การวนลูปและเปรียบเทียบสมาชิกแต่ละตัว เพื่อหาค่าที่สูงที่สุด ตัวอย่างนี้แสดงให้เห็นการประยุกต์ใช้ลูปและเงื่อนไขเพื่อดึงข้อมูลสำคัญออกจากลิสต์

ตัวอย่าง:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

def find_largest(numbers):
    largest = numbers[0]
    for number in numbers:
        if number > largest:
            largest = number
    return largest

print(f"The largest number is {find_largest(numbers)}")
# Output: The largest number is 9
```

Listing 6.34: Finding the Largest Number in a List

6.7.2 ตัวอย่าง: การลบค่าซ้ำในลิสต์

การลบค่าซ้ำออกจากลิสต์ทำได้โดยการวนลูปตรวจสอบแต่ละค่า และเพิ่มเฉพาะค่าที่ไม่ซ้ำลงในลิสต์ใหม่ วิธีนี้ช่วยให้ได้ลิสต์ที่ไม่มีค่าซ้ำ แสดงให้เห็นถึงการใช้ลูปและเงื่อนไขในการจัดการข้อมูลให้มีความสมบูรณ์

ตัวอย่าง:

```
def remove_duplicates(numbers):
    unique_numbers = []
    for number in numbers:
        if number not in unique_numbers:
            unique_numbers.append(number)
    return unique_numbers

numbers = [1, 2, 3, 1, 2, 4, 5, 6, 5, 4, 3]
print(f"Original list: {numbers}")
print(f"List after removing duplicates: {remove_duplicates(numbers)}")
```

Listing 6.35: Removing Duplicates from a List

6.7.3 ตัวอย่าง: List Comprehension

List comprehension คือวิธีการสร้างลิสต์ในรูปแบบกระชับในภาษา Python โดยใช้การวนซ้ำร่วมกับนิพจน์ในบรรทัดเดียว ทำให้โค้ดอ่านง่ายและมีประสิทธิภาพมากขึ้น ตัวอย่างเช่น `[x**2 for x in range(10)]` จะสร้างลิสต์ที่มีเลขยกกำลังสองตั้งแต่ 0 ถึง 9 เหมาะสำหรับการแปลงข้อมูล การกรอง และการคำนวณในลิสต์อย่างรวดเร็ว

ตัวอย่าง:

```
# Creating a list of squares using list comprehension
squares = [x**2 for x in range(10)]
print(squares)  # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Listing 6.36: Example: List Comprehension

ตัวอย่าง:

```
import timeit

# Define the function you want to time
def create_squares():
    squares = [x**2 for x in range(100000)]

# Use timeit to measure the execution time of the function
execution_time = timeit.timeit(create_squares, number=1000)
print(f"Execution time: {execution_time} seconds")
```

Listing 6.37: Example: TimeIt List Comprehension

ตัวอย่าง:

```
import timeit

# Define the function you want to time
def create_squares():
    squares = []
    for i in range(100000):
        squares.append(i)

# Use timeit to measure the execution time of the function
execution_time = timeit.timeit(create_squares, number=1000)
print(f"Execution time: {execution_time} seconds")
```

Listing 6.38: Example: TimeIt List ComprehensionII

บทที่ 6 โจทย์และแบบฝึกหัด: ลิสต์และทูเพิล

6.1 การสร้างและเข้าถึงลิสต์
สร้างลิสต์ที่ประกอบด้วยเลขฟีโบนัชชี 15 ตัวแรก และเข้าถึงสมาชิกตัวที่ 10

6.2 การใช้ดัชนีติดลบ
จากลิสต์ `months = ["January", ..., "December"]` ให้เข้าถึง 3 เดือนสุดท้ายโดยใช้ดัชนีติดลบ

6.3 การแก้ไขลิสต์
แทนที่สมาชิกตำแหน่งที่ 2 และ 4 ในลิสต์ `shapes` ด้วย "ellipse" และ "pentagon"

6.4 เมธอด `append()`
ใช้ลูปเพื่อเพิ่มสมาชิกจากลิสต์ `more_fruits` ไปยัง `fruits`

6.5 เมธอด `insert()`
แทรก "strawberry" และ "blueberry" ลงในตำแหน่งที่ 2 และ 3 ของลิสต์ `berries`

6.6 เมธอด `remove()`
ลบทุกค่าที่เป็น "apple" ออกจากลิสต์ `fruits`

6.7 เมธอด `pop()`
ลบและคืนค่าของสมาชิกตัวที่ 3 จากลิสต์ `grades` แล้วเพิ่มค่านั้นไว้ที่ท้ายลิสต์

6.8 เมธอด `clear()`
ล้างข้อมูลทั้งหมดใน `nested_list` โดยคงโครงสร้างลิสต์ชั้นนอกไว้

6.9 การใช้ Slicing กับลิสต์
จากลิสต์ `data = list(range(100))` ให้ตัดลิสต์โดยเลือกทุก ๆ 5 ตัว เริ่มจากสมาชิกตัวที่ 10 ถึงตัวที่ 50

6.10 หาค่ามากที่สุดจากลิสต์ซ้อน
เขียนฟังก์ชันเพื่อหาค่ามากที่สุดจาก `nested_numbers = [[10, 25, 30], [45, 22], [75, 40, 100]]`

ภาคผนวก

ภาคผนวกนี้นำเสนอแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งาน เพื่อเสริมความเข้าใจในเนื้อหาที่ครอบคลุมในบทที่ 6

A6.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Python for Data Analysis* เขียนโดย Wes McKinney
- *Python Programming: An Introduction to Computer Science* เขียนโดย John Zelle

บทเรียนออนไลน์:

- Python Lists - W3Schools
- Lists in Python - Real Python

คอร์สเรียน:

- Coursera: Data Collection and Processing with Python โดยมหาวิทยาลัย Michigan
- Udacity: Introduction to Python Programming

A6.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). Python Documentation - Data Structures. สืบค้นจาก https://docs.python.org/3/tutorial/datastructures.html
- Sweigart, A. (2015). *Automate the Boring Stuff with Python*. สำนักพิมพ์ No Starch Press.

A6.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การดำเนินการเบื้องต้นกับลิสต์

- เขียนโปรแกรม Python เพื่อสร้างลิสต์รายการอาหารที่ชอบและแสดงผลทีละรายการ
- สร้างลิสต์ตัวเลขจาก 1 ถึง 10 และหาผลรวมของตัวเลขในลิสต์

แบบฝึกหัดที่ 2: เมธอดของลิสต์

- พัฒนาฟังก์ชันที่รับลิสต์ตัวเลขและส่งกลับลิสต์ใหม่ที่เรียงลำดับจากน้อยไปมาก
- เขียนสคริปต์ Python เพื่อลบค่าซ้ำจากลิสต์ของตัวเลข

แบบฝึกหัดที่ 3: การดำเนินการขั้นสูงกับลิสต์

- เขียนโปรแกรมที่รับลิสต์ของคำและส่งคืนคำที่ยาวที่สุด
- เขียนฟังก์ชันที่รับลิสต์ตัวเลขและส่งกลับลิสต์ที่มีเฉพาะเลขคู่เท่านั้น

แบบฝึกหัดที่ 4: การใช้ List Comprehensions

- ใช้ list comprehension เพื่อสร้างลิสต์ของเลขยกกำลังสองจาก 1 ถึง 10
- เขียน list comprehension เพื่อกรองเลขคี่ออกจากลิสต์ของจำนวนเต็ม

A6.4 ตัวอย่างการใช้งาน

ตัวอย่างที่ 1: การดำเนินการเบื้องต้นกับลิสต์

```
# List of favorite foods
favorite_foods = ["Pizza", "Burger", "Pasta", "Sushi", "Ice Cream"]
for food in favorite_foods:
    print(food)
```

Listing 6.39: List of favorite foods

ตัวอย่างที่ 2: การใช้เมธอดของลิสต์

```
# Function to remove duplicates from a list
def remove_duplicates(numbers):
    return list(set(numbers))

# Test the function
numbers = [1, 2, 2, 3, 4, 4, 5]
print(remove_duplicates(numbers))  # Output: [1, 2, 3, 4, 5]
```

Listing 6.40: Function to remove duplicates from a list

ตัวอย่างที่ 3: การดำเนินการขั้นสูงกับลิสต์

```
# Function to find the longest word in a list
def longest_word(words):
    return max(words, key=len)

# Test the function
words = ["apple", "banana", "cherry", "date"]
print(longest_word(words))  # Output: "banana"
```

Listing 6.41: Function to find the longest word in a list

ตัวอย่างที่ 4: การใช้ List Comprehensions

```
# List comprehension to create a list of squares of numbers from 1 to 10
squares = [x ** 2 for x in range(1, 11)]
print(squares)  # Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

# List comprehension to filter out odd numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = [x for x in numbers if x % 2 == 0]
print(evens)  # Output: [2, 4, 6, 8, 10]
```

Listing 6.42: List comprehensions for squares and filtering odd numbers

บทที่ 7

เซต ดิกชันนารี และการดำเนินการพื้นฐาน

บทนี้กล่าวถึงโครงสร้างข้อมูลพื้นฐานสองประเภทในภาษา Python ได้แก่ เซต (Set) และดิกชันนารี (Dictionary) ซึ่งเป็นเครื่องมือที่สำคัญสำหรับการจัดการข้อมูล การประมวลผล และการค้นคืนข้อมูลอย่างมีประสิทธิภาพ โดยเซตเป็นกลุ่มของข้อมูลที่ไม่เรียงลำดับและไม่มีการซ้ำกัน เหมาะสำหรับงานที่ต้องการลบค่าซ้ำ การตรวจสอบสมาชิก และการดำเนินการเชิงคณิตศาสตร์ เช่น ยูเนียน อินเทอร์เซกชัน และดิฟเฟอเรนซ์ ส่วนดิกชันนารีเก็บข้อมูลในรูปแบบคู่คีย์และค่า (key-value pairs) ซึ่งช่วยให้สามารถค้นหา อัปเดต และจัดการข้อมูลได้อย่างรวดเร็วและยืดหยุ่น

ตัวอย่างจริงที่แสดงในบทนี้จะช่วยให้ผู้อ่านเข้าใจการประยุกต์ใช้เซตในการกรองค่าที่ไม่ซ้ำกันและการดำเนินการทางเซต ขณะเดียวกัน ดิกชันนารีก็โดดเด่นในสถานการณ์ที่ต้องการจัดเก็บข้อมูลแบบไดนามิก การจัดกลุ่ม และการค้นหาจากคีย์อย่างรวดเร็ว เนื้อหาครอบคลุมนี้จะช่วยให้ผู้อ่านสามารถใช้เครื่องมืออันทรงพลังเหล่านี้ในงานต่าง ๆ ของการเขียนโปรแกรมด้วย Python ตั้งแต่การจัดการข้อมูลพื้นฐานไปจนถึงโครงสร้างข้อมูลขั้นสูงและอัลกอริธึมที่ซับซ้อน

7.1 แนะนำเซต (Set)

เซตคือกลุ่มของข้อมูลที่ไม่เรียงลำดับและไม่ซ้ำกันในภาษา Python หมายความว่า ข้อมูลแต่ละรายการในเซตจะไม่มีลำดับแน่นอน และไม่สามารถมีข้อมูลซ้ำกันได้ โครงสร้างข้อมูลประเภทนี้เหมาะสำหรับเก็บข้อมูลหลายค่าไว้ในตัวแปรเดียวโดยรับประกันว่าทุกค่าจะไม่ซ้ำกัน เมื่อเพิ่มข้อมูลลงในเซต ระบบจะลบค่าที่ซ้ำโดยอัตโนมัติ ทำให้เซตเหมาะสำหรับงานที่เกี่ยวข้องกับข้อมูลที่ไม่ซ้ำกัน เช่น การตรวจสอบสมาชิก การลบค่าซ้ำ และการดำเนินการทางคณิตศาสตร์อย่าง ยูเนียน อินเทอร์เซกชัน และดิฟเฟอเรนซ์ เซตสามารถสร้างได้โดยใช้เครื่องหมายปีกกา `{}` หรือฟังก์ชัน `set()`

แนวคิดสำคัญของเซต:

- **กลุ่มข้อมูลที่ไม่เรียงลำดับ**: เซตไม่รักษาลำดับของข้อมูลที่เก็บอยู่
- **ข้อมูลไม่ซ้ำกัน**: เซตจะลบค่าที่ซ้ำโดยอัตโนมัติ
- **ปรับเปลี่ยนได้**: เซตสามารถเพิ่มหรือลบข้อมูลได้ภายหลัง
- **ตรวจสอบสมาชิกได้ง่าย**: มีฟังก์ชันสำหรับตรวจสอบว่าสมาชิกอยู่ในเซตหรือไม่
- **การดำเนินการเชิงคณิตศาสตร์**: รองรับยูเนียน อินเทอร์เซกชัน และดิฟเฟอเรนซ์
- **ไม่รองรับการเข้าถึงด้วยดัชนี**: ไม่สามารถเข้าถึงข้อมูลด้วยดัชนีหรือตัดช่วงได้เหมือนลิสต์
- **ขนาดเปลี่ยนแปลงได้**: จำนวนสมาชิกในเซตสามารถเปลี่ยนแปลงได้
- **สมาชิกต้องแฮชได้**: สมาชิกของเซตต้องเป็นชนิดข้อมูลที่ไม่เปลี่ยนแปลง เช่น string, number, หรือ tuple
- **การใช้งาน**: ใช้เมื่อต้องการข้อมูลที่ไม่ซ้ำกัน การลบค่าซ้ำ และการดำเนินการทางเซต
- **การสร้าง**: สร้างได้ด้วย `{}` หรือ `set()` จากลิสต์หรือทูเพิล

7.1.1 การสร้างและเข้าถึงเซต

สามารถสร้างเซตได้โดยใส่ค่าคั่นด้วยเครื่องหมายจุลภาคภายในวงเล็บปีกกา `{}` ซึ่งสามารถกำหนดเซตที่มีหลายค่าที่ไม่ซ้ำกันในรูปแบบที่กระชับ อีกทางเลือกหนึ่งคือการใช้ฟังก์ชัน `set()` ซึ่งมีประโยชน์เมื่อต้องการแปลงอ็อบเจกต์แบบวนซ้ำ เช่น ลิสต์หรือทูเพิล เป็นเซต โดยฟังก์ชันนี้จะลบค่าซ้ำออก ทำให้ได้กลุ่มของค่าที่ไม่ซ้ำกัน วิธีเหล่านี้ช่วยให้สามารถกำหนดเซตได้อย่างรวดเร็วและมีประสิทธิภาพ เพื่อรองรับการตรวจสอบสมาชิก การลบค่าซ้ำ และการดำเนินการทางคณิตศาสตร์แบบเซต

```python
# Creating a set
fruits = {"apple", "banana", "cherry", "apple"}
print(fruits)  # Output: {'apple', 'banana', 'cherry'} (duplicates are
    removed)

# Creating a set using the set() function
numbers = set([1, 2, 3, 1, 2, 4])
print(numbers)  # Output: {1, 2, 3, 4}
```

Listing 7.1: Creating and Accessing Sets

7.1.2 การจัดการสมาชิกของเซต

การเพิ่มหรือลบข้อมูลในเซตสามารถทำได้อย่างตรงไปตรงมา ซึ่งช่วยเพิ่มความยืดหยุ่นและประโยชน์ของเซต โดยสามารถเพิ่มข้อมูลใหม่ได้โดยมั่นใจว่าจะไม่มีข้อมูลซ้ำ และสามารถลบข้อมูลออกได้อย่างง่ายดาย พร้อมตัวเลือกในการลบทั้งหมดหรือบางรายการ วิธีการเหล่านี้ทำให้เซตเหมาะสมสำหรับการใช้งานที่ต้องปรับปรุงข้อมูลบ่อย เช่น การจัดการคอลเลกชันที่ไม่ซ้ำกัน การตรวจสอบสมาชิก และการกรองหรือทำความสะอาดข้อมูล

ฟังก์ชันสำหรับการจัดการสมาชิกในเซต

- `add()`: เพิ่มสมาชิกหนึ่งรายการ
- `update()`: เพิ่มหลายรายการพร้อมกัน
- `remove()`: ลบสมาชิก หากไม่พบจะเกิดข้อผิดพลาด
- `discard()`: ลบสมาชิก หากไม่พบจะไม่เกิดข้อผิดพลาด
- `pop()`: ลบและคืนค่ารายการหนึ่งแบบสุ่ม
- `clear()`: ลบสมาชิกทั้งหมดในเซต

```python
# Initial set of fruits
fruits = {"apple", "banana", "cherry"}

# Adding a single item
fruits.add("orange")
print(fruits)  # Output: {'apple', 'banana', 'cherry', 'orange'}

# Adding multiple items using update()
fruits.update(["mango", "grape"])
print(fruits)  # Output: {'apple', 'banana', 'cherry', 'orange', 'mango', '
    grape'}

# Removing an item
fruits.remove("banana")
print(fruits)  # Output: {'apple', 'cherry', 'orange', 'mango', 'grape'}

# Discarding an item (no error if the item doesn't exist)
fruits.discard("pineapple")
print(fruits)  # Output: {'apple', 'cherry', 'orange', 'mango', 'grape'} (
    no error raised)

# Removing and returning an arbitrary item
removed_item = fruits.pop()
print(removed_item)  # Output: (varies, since pop removes an arbitrary item
    )
print(fruits)  # Output: (remaining items after pop)

# Clearing the set
fruits.clear()
print(fruits)  # Output: set() (empty set)
```

Listing 7.2: Adding and Removing Items to Set

7.1.3 การดำเนินการของเซต

การดำเนินการของเซตรวมถึงการเปรียบเทียบและรวมเซตในรูปแบบต่าง ๆ ที่มีประสิทธิภาพ เช่น ยูเนียน (รวมสมาชิกทั้งหมด), อินเทอร์เซกชัน (หาสมาชิกที่ซ้ำกัน), ดิฟเฟอเรนซ์ (หาสมาชิกที่มีในเซตแรกแต่ไม่มีในเซตที่สอง) และสมมาตรดิฟเฟอเรนซ์ (สมาชิกที่มีอยู่ในเซตใดเซตหนึ่ง แต่ไม่ใช่ทั้งสอง) การดำเนินการเหล่านี้ช่วยให้สามารถประมวลผลข้อมูลที่แตกต่างกันได้อย่างรวดเร็วและแม่นยำ

ฟังก์ชันและสัญลักษณ์ของการดำเนินการเซต

- `union()` `|`: รวมสมาชิกของทั้งสองเซต
- `intersection()` `&`: หาสมาชิกที่มีร่วมกัน
- `difference()` `—`: หาสมาชิกที่มีในเซตแรกแต่ไม่มีในเซตที่สอง
- `symmetric_difference()^` : หาสมาชิกที่อยู่ในเซตใดเซตหนึ่งเท่านั้น

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# Union
print(set1.union(set2))  # Output: {1, 2, 3, 4, 5}

# Intersection
print(set1.intersection(set2))  # Output: {3}

# Difference
print(set1.difference(set2))  # Output: {1, 2}

# Symmetric Difference
print(set1.symmetric_difference(set2))  # Output: {1, 2, 4, 5}
```

Listing 7.3: Set Operations

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

# Union
union_set = set1 | set2
print("Union:", union_set)  # Output: {1, 2, 3, 4, 5, 6}

# Intersection
intersection_set = set1 & set2
print("Intersection:", intersection_set)  # Output: {3, 4}

# Difference
difference_set = set1 - set2
print("Difference:", difference_set)  # Output: {1, 2}

# Symmetric Difference
sym_diff_set = set1 ^ set2
print("Symmetric Difference:", sym_diff_set)
# Output: {1, 2, 5, 6}
```

Listing 7.4: Set Symbol Operations

7.1.4 การอัปเดตการดำเนินการของเซต

ในภาษา Python เราสามารถอัปเดตเซตได้โดยตรงด้วยการใช้โอเปอเรเตอร์พิเศษ ซึ่งจะปรับปรุงค่าในเซตโดยอิงจากค่าของเซตอื่น โดยโอเปอเรเตอร์เหล่านี้ได้แก่ **&=**, **-=**, และ **≙** ซึ่งมีความหมายดังนี้:

- **&=** (Intersection Update): เก็บไว้เฉพาะสมาชิกที่มีอยู่ในทั้งสองเซต
- **-=** (Difference Update): ลบสมาชิกที่มีอยู่ในเซตอีกชุดออกจากเซตเดิม
- **≙** (Symmetric Difference Update): เก็บไว้เฉพาะสมาชิกที่อยู่ในเซตใดเซตหนึ่งเท่านั้น แต่อยู่ในทั้งสองไม่ได้

ตัวอย่างด้านล่างแสดงการใช้โอเปอเรเตอร์เหล่านี้:

```
# Initial sets
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7}

# &= Intersection Update: set1 will be updated only to include elements present in both sets
set1 &= set2
print("After &= operation:", set1)  # Output: {4, 5}

# Resetting set1 to its original value
set1 = {1, 2, 3, 4, 5}

# -= Difference Update: set1 will be updated to remove elements also present in set2
set1 -= set2
print("After -= operation:", set1)  # Output: {1, 2, 3}

# Resetting set1 to its original value
set1 = {1, 2, 3, 4, 5}

# ^= Symmetric Difference Update: set1 will be updated to keep elements in either set but not in both
set1 ^= set2
print("After ^= operation:", set1)  # Output: {1, 2, 3, 6, 7}
```

Listing 7.5: Demonstration of **&= -= ≙** operations in Python sets

คำอธิบาย

- **Intersection Update (&=)**: `set1` จะถูกอัปเดตให้มีเฉพาะ `{4, 5}` ซึ่งเป็นสมาชิกที่มีอยู่ในทั้ง `set1` และ `set2`
- **Difference Update (-=)**: `set1` จะถูกอัปเดตเป็น `{1, 2, 3}` โดยลบ `{4, 5}` ที่มีใน `set2` ออกไป
- **Symmetric Difference Update (≙)**: `set1` จะถูกอัปเดตให้เหลือ `{1, 2, 3, 6, 7}` ซึ่งเป็นสมาชิกที่ไม่ซ้ำกันในแต่ละเซต

7.1.5 การดำเนินการของเซต: ซับเซตและซูเปอร์เซต

โอเปอเรเตอร์เหล่านี้ใช้สำหรับเปรียบเทียบความสัมพันธ์ระหว่างเซตสองชุดว่ามีลักษณะเป็นซับเซต ซูเปอร์เซต หรือเท่ากัน ดังนี้:

- **>=** (ซูเปอร์เซต): ตรวจสอบว่าเซตหนึ่งเป็นซูเปอร์เซตของอีกเซตหรือไม่ (มีสมาชิกทั้งหมดของอีกเซต)

- <= (ซับเซต): ตรวจสอบว่าเซตหนึ่งเป็นซับเซตของอีกเซตหรือไม่ (สมาชิกทั้งหมดอยู่ในอีกเซต)
- > (ซูเปอร์เซตแท้): เป็นซูเปอร์เซตและมีสมาชิกมากกว่าอีกเซต
- < (ซับเซตแท้): เป็นซับเซตและมีสมาชิกน้อยกว่าอีกเซต
- == (เท่ากัน): ตรวจสอบว่าเซตสองชุดมีสมาชิกเท่ากันทุกตัว

```python
# Define two sets
set_a = {1, 2, 3, 4}
set_b = {2, 3}
set_c = {1, 2, 3, 4}
set_d = {1, 2, 3, 4, 5}

# Superset and Subset
print("Is set_a a superset of set_b?:", set_a >= set_b)  # Output: True
print("Is set_b a subset of set_a?:", set_b <= set_a)    # Output: True

# Proper Superset and Proper Subset
print("Is set_a a proper superset of set_b?:", set_a > set_b)  # Output:
    True
print("Is set_b a proper subset of set_a?:", set_b < set_a)    # Output:
    True

# Equal Sets
print("Are set_a and set_c equal?:", set_a == set_c)  # Output: True

# Related but not equal (one is a subset but not equal)
print("Is set_b a subset of set_d and not equal?:", set_b <= set_d and
    set_b != set_d)  # Output: True
```

Listing 7.6: Demonstration of Set Operations in Python

คำอธิบาย

- **ซูเปอร์เซต (>=)**: `set_a >= set_b` หมายถึง `set_a` มีสมาชิกครบทุกตัวที่อยู่ใน `set_b` คำตอบคือ True
- **ซับเซต (<=)**: `set_b <= set_a` หมายถึงสมาชิกใน `set_b` อยู่ใน `set_a` ทั้งหมด คำตอบคือ True
- **ซูเปอร์เซตแท้ (>)**: `set_a > set_b` หมายถึง `set_a` เป็นซูเปอร์เซตของ `set_b` และมีสมาชิกมากกว่า คำตอบคือ True
- **ซับเซตแท้ (<)**: `set_b < set_a` หมายถึง `set_b` เป็นซับเซตของ `set_a` และมีสมาชิกน้อยกว่า คำตอบคือ True
- **เซตที่เท่ากัน (==)**: `set_a == set_c` หมายถึงทั้งสองเซตมีสมาชิกเท่ากัน คำตอบคือ True
- **เกี่ยวข้องแต่ไม่เท่ากัน**: `set_b <= set_d and set_b != set_d` ตรวจสอบว่า `set_b` เป็นซับเซตของ `set_d` และไม่เท่ากัน ซึ่งคำตอบคือ True

ตัวอย่างนี้แสดงให้เห็นว่าเราสามารถใช้โอเปอเรเตอร์ของเซตเพื่อเปรียบเทียบความสัมพันธ์ เช่น ซับเซต ซูเปอร์เซต หรือเท่ากันได้อย่างมีประสิทธิภาพ

7.2 บทนำสู่

Dictionary คือโครงสร้างข้อมูลที่ไม่เรียงลำดับ (unordered) ประกอบด้วยคู่ของ **key-value** ซึ่ง `key` แต่ละตัวต้องไม่ซ้ำกัน และใช้เพื่อเข้าถึง `value` ที่เกี่ยวข้อง ในภาษา Python, dictionary มีประสิทธิภาพสูงในการเข้าถึง เพิ่ม และจัดการข้อมูล เนื่องจากใช้โครงสร้างข้อมูลแบบ *hash table* โดย `key` จะต้องเป็นชนิดข้อมูลที่ไม่เปลี่ยนแปลง (immutable) เช่น สตริง ตัวเลข หรือทูเพิล ส่วน `value` สามารถเป็นชนิดข้อมูลใดก็ได้ โครงสร้างนี้ช่วยให้สามารถจัดเก็บข้อมูลได้อย่างยืดหยุ่นและไดนามิก รวมถึงสามารถเข้าถึงและปรับปรุงข้อมูลได้อย่างรวดเร็ว เหมาะสำหรับกรณีที่ต้องการติดป้ายชื่อข้อมูลหรือรักษาความสัมพันธ์ระหว่างข้อมูลที่เป็นคู่กัน

แนวคิดหลักของ Dictionary:

- **ไม่เรียงลำดับ (Unordered Collection)**: ไม่มีลำดับที่แน่นอนของสมาชิก
- **คู่คีย์-ค่า (Key-Value Pairs)**: ข้อมูลถูกจัดเก็บเป็นคู่คีย์และค่าที่สัมพันธ์กัน
- **คีย์ต้องไม่ซ้ำกัน (Unique Keys)**: คีย์แต่ละตัวต้องไม่ซ้ำ
- **เปลี่ยนแปลงได้ (Mutable)**: สามารถแก้ไขหรือปรับเปลี่ยนได้หลังจากสร้างแล้ว
- **ขนาดเปลี่ยนแปลงได้ (Dynamic Size)**: เพิ่มหรือลบคู่คีย์-ค่าได้ตามต้องการ
- **เข้าถึงได้รวดเร็ว (Efficient Lookup)**: ค้นหาค่าได้อย่างรวดเร็วโดยใช้คีย์
- **คีย์ต้อง hash ได้ (Hashable Keys)**: คีย์ต้องเป็นชนิดข้อมูลที่ไม่เปลี่ยนแปลง
- **ค่ามีความยืดหยุ่น (Flexible Values)**: ค่าสามารถเป็นข้อมูลชนิดใดก็ได้
- **มีเมท็อดหลากหลาย (Comprehensive Methods)**: มีฟังก์ชันสำหรับเข้าถึง ตรวจสอบ และลบข้อมูล
- **การใช้งาน (Use Cases)**: เหมาะสำหรับจัดเก็บข้อมูลที่มีป้ายกำกับหรือต้องการเชื่อมโยงเป็นคู่

7.2.1 การสร้างและเข้าถึง Dictionary

Dictionary สร้างขึ้นโดยใช้วงเล็บปีกกา `{}` ซึ่งประกอบด้วยคู่คีย์และค่าที่คั่นด้วยเครื่องหมายโคลอน `:` โดยแต่ละคู่จะคั่นด้วยเครื่องหมายจุลภาค โครงสร้างนี้ช่วยให้สามารถจัดเก็บข้อมูลในรูปแบบที่อ้างอิงด้วยคีย์ได้อย่างชัดเจน และเข้าถึงหรือแก้ไขข้อมูลได้อย่างมีประสิทธิภาพ

1. **การสร้าง Dictionary โดยใช้ {}** เป็นวิธีที่พบได้บ่อยที่สุด โดยกำหนดคู่คีย์และค่าภายในวงเล็บปีกกา {} และใช้ `:` เชื่อมระหว่างคีย์กับค่า

```python
# Creating a dictionary using {}
student = {"name": "Alice", "age": 25, "grade": "A"}
print(student)  # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.7: การสร้าง Dictionary โดยใช้ {}

2. การสร้าง Dictionary โดยใช้ `dict()` สามารถสร้าง dictionary โดยใช้ constructor `dict()` เหมาะกับกรณีที่ต้องการสร้างจากอาร์กิวเมนต์แบบคีย์เวิร์ดหรือลิสต์ของทูเพิล

```
# Creating a dictionary using dict() with keyword arguments
student = dict(name="Alice", age=25, grade="A")
print(student)  # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}

# Creating a dictionary using dict() with a list of tuples
student = dict([("name", "Alice"), ("age", 25), ("grade", "A")])
print(student)  # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.8: การสร้าง Dictionary โดยใช้ `dict()` และคีย์เวิร์ดอาร์กิวเมนต์

3. การสร้าง Dictionary ด้วย = วิธีนี้เริ่มจากกำหนดตัวแปรให้เป็น dictionary ว่าง แล้วเพิ่มคู่คีย์-ค่าทีละรายการโดยใช้เครื่องหมายเท่ากับ =

```
# Creating an empty dictionary
student = {}

# Adding key-value pairs using assignment
student["name"] = "Alice"
student["age"] = 25
student["grade"] = "A"
print(student)  # Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.9: การสร้าง Dictionary ว่างและเพิ่มข้อมูลด้วย =

- **ใช้ {}**: สร้าง dictionary โดยตรงจากคู่คีย์-ค่า
- **ใช้ `dict()`**: สร้าง dictionary จากคีย์เวิร์ดอาร์กิวเมนต์ หรือจากทูเพิล
- **ใช้ =**: สร้างแบบว่างก่อน แล้วเพิ่มคีย์-ค่าภายหลัง

```
# Creating a dictionary
student = {"name": "Alice", "age": 25, "grade": "A"}

# Accessing values by key
print(student["name"])  # Output: Alice
print(student["age"])   # Output: 25
print(student["grade"]) # Output: A
```

Listing 7.10: การสร้างและเข้าถึงข้อมูลใน Dictionary

7.2.2 การเข้าถึง Dictionary ด้วยการวนลูป

การเข้าถึงข้อมูลใน Dictionary ด้วยการใช้ลูปเป็นวิธีที่มีประสิทธิภาพสำหรับการวนผ่านคีย์ ค่า หรือคู่คีย์-ค่า โดยค่าเริ่มต้นเมื่อลูปผ่าน Dictionary จะวนผ่านคีย์ ซึ่งสามารถใช้เข้าถึงค่าที่เกี่ยวข้องได้โดยตรง ฟังก์ชัน `.values()` ใช้เพื่อวนผ่านเฉพาะค่า และ `.items()` ช่วยให้สามารถเข้าถึงทั้งคีย์และค่าได้พร้อมกัน ความยืดหยุ่นนี้ทำให้ลูปเหมาะสำหรับงานประมวลผล ปรับเปลี่ยน และวิเคราะห์ข้อมูลภายใน Dictionary ลูปสามารถใช้ปรับปรุงค่า คัดกรองข้อมูล หรือรวมผลลัพธ์ ซึ่งเป็นเครื่องมือที่ทรงพลังในการจัดการ Dictionary ด้วยภาษา Python

ตัวอย่างที่ 1: วนผ่านคีย์และเข้าถึงค่า ตัวอย่างนี้แสดงการวนผ่านคีย์และเข้าถึงค่าที่เกี่ยวข้อง

```
student = {"name": "Alice", "age": 25, "grade": "A", "major": "Computer
    Science"}

for key in student:
    print(f"{key}: {student[key]}")
# Output:
# name: Alice
# age: 25
# grade: A
# major: Computer Science
```

Listing 7.11: วนผ่านคีย์และเข้าถึงค่า

ตัวอย่างที่ 2: วนผ่านเฉพาะค่า ตัวอย่างนี้แสดงวิธีวนลูปผ่านเฉพาะค่าของ Dictionary

```
student = {"name": "Alice", "age": 25, "grade": "A", "major": "Computer
    Science"}

for value in student.values():
    print(value)
# Output:
# Alice
# 25
# A
# Computer Science
```

Listing 7.12: วนผ่านเฉพาะค่า

ตัวอย่างที่ 3: วนผ่านคู่คีย์-ค่า ตัวอย่างนี้แสดงการใช้เมท็อด `.items()` เพื่อวนผ่านทั้งคีย์และค่าพร้อมกัน

```
student = {"name": "Alice", "age": 25, "grade": "A", "major": "Computer
    Science"}

for key, value in student.items():
    print(f"{key}: {value}")
# Output:
# name: Alice
# age: 25
# grade: A
# major: Computer Science
```

Listing 7.13: วนผ่านคู่คีย์-ค่า

ตัวอย่างเหล่านี้แสดงวิธีที่แตกต่างในการเข้าถึงและจัดการข้อมูลใน Dictionary ด้วยการใช้ลูป ซึ่งช่วยให้การประมวลผลข้อมูลมีความยืดหยุ่นและมีประสิทธิภาพใน Python

7.2.3 การปรับปรุง Dictionary

การเพิ่ม ปรับปรุง หรือลบคู่คีย์-ค่าใน Dictionary เป็นการจัดการข้อมูลที่ยืดหยุ่นสูง การเพิ่มหรือปรับปรุงสามารถทำได้โดยการกำหนดค่าให้กับคีย์ ในขณะที่การลบสามารถทำได้โดยใช้คำสั่ง `del` หรือเมท็อด `pop()` ลักษณะ

ไดนามิกนี้ทำให้ Dictionary สามารถปรับตัวให้เข้ากับความต้องการของข้อมูลที่เปลี่ยนแปลงได้ เหมาะสำหรับการจัดเก็บข้อมูล การเข้าถึง และการอัปเดตแบบเรียลไทม์

```
# Creating a dictionary
student = {"name": "Alice", "age": 25, "grade": "A"}

# Adding and updating values
student["age"] = 26
student["major"] = "Computer Science"
print(student)  # Output: {'name': 'Alice', 'age': 26, 'grade': 'A', 'major
    ': 'Computer Science'}

# Removing a key-value pair using del
del student["grade"]
print(student)  # Output: {'name': 'Alice', 'age': 26, 'major': 'Computer
    Science'}

# Removing a key-value pair using pop and getting the removed value
removed_major = student.pop("major")
print(removed_major)  # Output: 'Computer Science'
print(student)  # Output: {'name': 'Alice', 'age': 26}
```

Listing 7.14: ตัวอย่าง: การใช้ `del` และ `pop()` ใน Dictionary

- **การใช้ `del`:** คำสั่ง `del` ใช้เพื่อลบคู่คีย์-ค่าที่มีคีย์เป็น `"grade"` ออกจาก Dictionary `student`
- **การใช้ `pop()`:** เมท็อด `pop()` ใช้ลบคู่คีย์-ค่าที่มีคีย์เป็น `"major"` และคืนค่าที่ถูกลบคือ `"Computer Science"` Dictionary `student` จะถูกอัปเดตโดยไม่มี entry ของ `"major"` อีกต่อไป

7.2.4 เมท็อดของ Dictionary

เมท็อดของ Dictionary มีเครื่องมือหลากหลายสำหรับจัดการและปรับเปลี่ยนข้อมูลที่เก็บอยู่ใน Dictionary เมท็อดเหล่านี้สามารถใช้เพื่อดึงคีย์ทั้งหมดหรือค่าทั้งหมด ตรวจสอบการมีอยู่ของคีย์ และดึงคู่คีย์-ค่าได้ นอกจากนี้ยังสามารถใช้เพิ่มหรือปรับปรุงรายการ ลบรายการเฉพาะ และล้างข้อมูลทั้งหมดใน Dictionary ได้อีกด้วย ความสามารถเหล่านี้ทำให้ Dictionary มีความยืดหยุ่นสูง เหมาะกับการใช้งานที่ต้องจัดการข้อมูลแบบไดนามิกและเรียกค้นข้อมูลอย่างมีประสิทธิภาพ

เมท็อดของ Dictionary

- `keys()`: คืนค่าออบเจ็กต์ที่เป็นมุมมองของคีย์ทั้งหมดใน Dictionary
- `values()`: คืนค่าออบเจ็กต์ที่เป็นมุมมองของค่าทั้งหมดใน Dictionary
- `items()`: คืนค่าออบเจ็กต์ที่เป็นมุมมองของคู่คีย์-ค่าทั้งหมดใน Dictionary
- `get()`: คืนค่าที่ตรงกับคีย์ที่ระบุ ถ้าคีย์นั้นมีอยู่ใน Dictionary
- `pop()`: ลบและคืนค่าที่ตรงกับคีย์ที่ระบุ
- `popitem()`: ลบและคืนคู่คีย์-ค่าล่าสุดจาก Dictionary
- `clear()`: ลบคู่คีย์-ค่าทั้งหมดใน Dictionary

```python
# Creating a dictionary
student = {'name': 'Alice', 'age': 26, 'major': 'Computer Science'}

# Dictionary methods
print(student.keys())   # Output: dict_keys(['name', 'age', 'major'])
print(student.values()) # Output: dict_values(['Alice', 26, 'Computer
    Science'])
print(student.items())  # Output: dict_items([('name', 'Alice'), ('age',
    26), ('major', 'Computer Science')])

# Using get() method
print(student.get("name"))  # Output: Alice
print(student.get("grade", "Not Found"))  # Output: Not Found

# Using pop() method
major = student.pop("major")
print(major)  # Output: Computer Science
print(student)  # Output: {'name': 'Alice', 'age': 26}

# Using popitem() method
last_item = student.popitem()
print(last_item)  # Output: ('age', 26)
print(student)  # Output: {'name': 'Alice'}

# Using clear() method
student.clear()
print(student)  # Output: {}
```

Listing 7.15: เมท็อดของ Dictionary

7.3 ตัวอย่างการใช้งานจริง

ตัวอย่างการใช้งานจริงในบทนี้แสดงให้เห็นถึงความสามารถอันทรงพลังของทูเพิล เซต และดิกชันนารีในภาษา Python ตัวอย่างประกอบด้วยการใช้ทูเพิลเป็นคีย์ของดิกชันนารีเพื่อจัดการข้อมูลซับซ้อน การใช้ดิกชันนารีในการนับจำนวนการเกิดของสมาชิกในลิสต์ การใช้เซตเพื่อกำจัดข้อมูลซ้ำอย่างมีประสิทธิภาพ และการแปลงลิสต์เป็นดิกชันนารีด้วย `zip()` สถานการณ์จริงเหล่านี้แสดงให้เห็นว่าการใช้โครงสร้างข้อมูลเหล่านี้สามารถทำให้งานจัดระเบียบข้อมูล การจัดการ และการวิเคราะห์เป็นเรื่องง่ายขึ้น ส่งผลให้ Python เป็นเครื่องมือที่หลากหลายสำหรับการเขียนโปรแกรม

7.3.1 ตัวอย่าง: การใช้ Tuple เป็นคีย์ใน Dictionary

การใช้ทูเพิลเป็นคีย์ในดิกชันนารีช่วยให้สามารถระบุคีย์ที่ซับซ้อนซึ่งแสดงถึงหลายคุณลักษณะได้ เทคนิคนี้เหมาะสำหรับสถานการณ์ที่ต้องใช้ค่าผสมที่ไม่ซ้ำกันเพื่อระบุ เช่น ค่าพิกัดในกริด หรือการค้นหาด้วยหลายคุณลักษณะ ซึ่งช่วยให้สามารถจัดระเบียบและเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพและชัดเจน

```
coordinates = {
    (0, 0): "Origin",
    (1, 0): "Point A",
    (0, 1): "Point B"
}
print(coordinates[(0, 0)])  # Output: Origin
```

Listing 7.16: Example: Tuple as Dictionary Keys

```
# Using tuples as keys in a dictionary
coordinates_info = {
    (13.7563, 100.5018): "Bangkok",
    (40.7128, -74.0060): "New York",
    (48.8566, 2.3522): "Paris"
}

# Accessing values using tuple keys
print(coordinates_info[(13.7563, 100.5018)])  # Output: Bangkok
print(coordinates_info[(40.7128, -74.0060)])  # Output: New York
print(coordinates_info[(48.8566, 2.3522)])    # Output: Paris
```

Listing 7.17: Example: Tuple as Dictionary Keys

7.3.2 ตัวอย่าง: การนับจำนวนสมาชิกในลิสต์ด้วย Dictionary

การใช้ดิกชันนารีเพื่อนับจำนวนการเกิดของแต่ละรายการในลิสต์เป็นวิธีที่มีประสิทธิภาพในการสรุปข้อมูล โดยใช้คีย์แทนสมาชิกแต่ละตัว และใช้ค่าของคีย์เพื่อนับจำนวน ทำให้สามารถวิเคราะห์และสรุปข้อมูลจากลิสต์ได้อย่างกระชับและชัดเจน

```
def count_elements(lst):
    counts = {}
    for item in lst:
        if item in counts:
            counts[item] += 1
        else:
            counts[item] = 1
    return counts

elements = ["apple", "banana", "apple", "cherry", "banana", "cherry", "
    cherry"]
print(count_elements(elements))
# Output: {'apple': 2, 'banana': 2, 'cherry': 3}
```

Listing 7.18: Counting Elements with a Dictionary

7.3.3 ตัวอย่าง: การลบค่าซ้ำในลิสต์ด้วย Set

การใช้เซตในการลบค่าซ้ำจากลิสต์เป็นวิธีที่มีประสิทธิภาพเพื่อให้แน่ใจว่าสมาชิกทุกตัวมีความไม่ซ้ำกัน เนื่องจากเซตสามารถเก็บได้เฉพาะค่าที่ไม่ซ้ำกัน การแปลงลิสต์เป็นเซตจะกรองค่าซ้ำออกโดยอัตโนมัติ วิธีนี้ทำให้การจัดการข้อมูลซ้ำเป็นไปอย่างง่ายและรวดเร็ว และยังสามารถแปลงกลับเป็นลิสต์เพื่อการประมวลผลเพิ่มเติมได้

```python
def remove_duplicates(lst):
    return list(set(lst))

numbers = [1, 2, 3, 1, 2, 4, 5, 6, 5, 4, 3]
print(remove_duplicates(numbers))
# Output: [1, 2, 3, 4, 5, 6]
```

Listing 7.19: Removing Duplicates from a List Using a Set

7.3.4 ตัวอย่าง: การแปลงลิสต์เป็น Dictionary

การใช้ `zip()` เพื่อแปลงลิสต์สองชุดให้เป็นดิกชันนารี เป็นวิธีที่มีประสิทธิภาพในการจับคู่ข้อมูล โดยลิสต์หนึ่งใช้เป็นคีย์และอีกลิสต์ใช้เป็นค่า ฟังก์ชัน `zip()` จะจับคู่ข้อมูลแล้วแปลงเป็นดิกชันนารีด้วย `dict()` ซึ่งเหมาะสำหรับการจัดระเบียบข้อมูลที่เกี่ยวข้องอย่างมีประสิทธิภาพ

```python
keys = ["name", "age", "grade"]
values = ["Alice", 25, "A"]
student = dict(zip(keys, values))
print(student)
# Output: {'name': 'Alice', 'age': 25, 'grade': 'A'}
```

Listing 7.20: Converting Lists to Dictionaries

7.3.5 ตัวอย่าง: พลังของการใช้ Set กับ List

คุณได้รับข้อมูลการเข้าเรียนของนักเรียนในห้องเรียนตลอดสัปดาห์ ซึ่งในแต่ละวันจะแสดงรายชื่อนักเรียนที่มาเรียนโดยอยู่ในรูปแบบลิสต์ เป้าหมายของคุณคือการวิเคราะห์ข้อมูลการเข้าเรียนโดยใช้เซตและลิสต์เพื่อตอบคำถามต่อไปนี้:

1. **หานักเรียนที่มาเรียนครบทุกวัน**
2. **หานักเรียนที่ขาดเรียนอย่างน้อยหนึ่งวัน**
3. **หานักเรียนที่มาเรียนวันแรกแต่ไม่มาเรียนวันสุดท้าย**
4. **คำนวณจำนวนนักเรียนทั้งหมดที่มาเรียนอย่างน้อยหนึ่งวัน**

ขั้นตอนการแก้ปัญหา:

1. **มาเรียนครบทุกวัน:** ใช้ `intersection()` ของเซตแต่ละวัน
2. **ขาดอย่างน้อยหนึ่งวัน:** หาผลต่างของ `union()` กับนักเรียนที่มาเรียนครบ
3. **มาเรียนวันแรกแต่ไม่มาเรียนวันสุดท้าย:** หาผลต่างของเซตวันแรกกับวันสุดท้าย
4. **จำนวนนักเรียนทั้งหมด:** ใช้ `union()` ของเซตทั้งหมด

```python
# Attendance records for a week (each list represents a day's attendance)
attendance_week = [
    ["Alice", "Bob", "Charlie", "David"],  # Day 1
    ["Alice", "Charlie", "David"],         # Day 2
    ["Alice", "Bob", "David"],             # Day 3
    ["Alice", "David", "Eve"],             # Day 4
    ["Bob", "Charlie", "David"]            # Day 5
]

# Convert each day's attendance list into a set
attendance_sets = [set(day) for day in attendance_week]

# 1. Find the set of students who were present every day.
present_every_day = set.intersection(*attendance_sets)
print("Present every day:", present_every_day)
# Output: {'David'}

# 2. Determine the set of students who were absent at least one day.
all_students = set.union(*attendance_sets)
absent_at_least_one_day = all_students - present_every_day
print("Absent at least one day:", absent_at_least_one_day)
# Output: {'Alice', 'Charlie', 'Bob', 'Eve'}

# 3. Create a list of students present on the first day but absent on the
    last day.
first_day_present = attendance_sets[0]
last_day_present = attendance_sets[-1]
first_day_but_not_last = list(first_day_present - last_day_present)
print("Present on first day but absent on last day:",
    first_day_but_not_last)
# Output: ['Alice']

# 4. Calculate the total number of unique students who attended at least
    one day.
unique_students_count = len(all_students)
print("Total unique students:", unique_students_count)
# Output: 5
```

Listing 7.21: Attendance records for a week

คำอธิบาย:

- **มาเรียนครบทุกวัน:** ใช้ `intersection` ของทุกเซตเพื่อหานักเรียนที่มาเรียนครบทุกวัน
- **ขาดอย่างน้อยหนึ่งวัน:** ใช้ `union` แล้วลบเซตของผู้มาเรียนครบ
- **มาเฉพาะวันแรกไม่มาเรียนวันสุดท้าย:** หาผลต่างของวันแรกและวันสุดท้าย
- **นักเรียนที่ไม่ซ้ำกันทั้งหมด:** ใช้ `union()` เพื่อรวมสมาชิกทั้งหมดแบบไม่ซ้ำกัน

บทที่ 7 โจทย์และแบบฝึกหัด: เซตและดิกชันนารี

7.1 สร้างและอัปเดตดิกชันนารีของนักเรียน
จากลิสต์ของทูเพิลที่เก็บชื่อ อายุ และเกรดของนักเรียน ให้สร้างดิกชันนารีที่ใช้ชื่อเป็นคีย์ และ (อายุ, เกรด) เป็นค่า แล้วเพิ่มอายุของนักเรียนแต่ละคนขึ้น 1 ปี

7.2 รวมดิกชันนารีหนังสือ
รวมดิกชันนารี 2 ชุดที่เก็บข้อมูลหนังสือ โดยรวมค่าที่ซ้ำในคีย์เดียวกันเป็นลิสต์

7.3 กรองจำนวนเฉพาะจากเซต
สร้างฟังก์ชันที่รับเซตของตัวเลข และคืนค่าเซตใหม่ที่มีเฉพาะจำนวนเฉพาะเท่านั้น

7.4 แปลงดิกชันนารีเกรดนักเรียน
จากดิกชันนารีที่เก็บชื่อนักเรียนและลิสต์เกรดของแต่ละคน ให้แปลงเป็นดิกชันนารีที่เก็บจำนวนเกรดและค่าเฉลี่ยเป็นทูเพิล

7.5 หาตำแหน่งของตัวอักษรในข้อความ
สร้างฟังก์ชันที่รับทูเพิลของข้อความ และคืนดิกชันนารีที่เก็บตัวอักษรไม่ซ้ำกันเป็นคีย์ และเซตของดัชนีข้อความที่ตัวอักษรนั้นปรากฏเป็นค่า

7.6 ดำเนินการกับเซต
เขียนฟังก์ชันที่รับเซต 2 ชุด แล้วคืนดิกชันนารีที่ประกอบด้วย union, intersection, difference และ symmetric_difference

7.7 คำนวณราคารวมของสินค้าในตะกร้า
สร้างฟังก์ชันที่รับดิกชันนารีราคาสินค้า และลิสต์ของสินค้าในตะกร้า พร้อมจำนวนที่ซื้อ เพื่อคำนวณราคารวมทั้งหมด

7.8 จัดกลุ่มคนตามช่วงอายุ
จากดิกชันนารีปีเกิดของแต่ละบุคคล ให้จำแนกเป็น 3 กลุ่ม: 'อายุไม่เกิน 18 ปี', '19 ถึง 35 ปี', และ '36 ปีขึ้นไป'

7.9 จัดกลุ่มประเทศตามจำนวนประชากร
จากเซตของทูเพิลที่เก็บชื่อประเทศและจำนวนประชากร ให้จัดประเภทประเทศเป็น 'เล็ก', 'กลาง', หรือ 'ใหญ่'

7.10 กลับด้านดิกชันนารีตำแหน่งพนักงาน
จากดิกชันนารีที่เก็บชื่อนักพนักงานและตำแหน่ง ให้กลับเป็นดิกชันนารีที่ใช้ตำแหน่งเป็นคีย์ และเซตของชื่อพนักงานเป็นค่า

ภาคผนวก

ภาคผนวกนี้นำเสนอแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งาน เพื่อเสริมความเข้าใจในเนื้อหาบทที่ 7 เกี่ยวกับ Tuple, Set และ Dictionary ในภาษา Python

ภาคผนวก A7.1 แหล่งเรียนรู้เพิ่มเติม

หนังสือ:

- *Python Programming: An Introduction to Computer Science* โดย John Zelle
- *Effective Python: 90 Specific Ways to Write Better Python* โดย Brett Slatkin

บทเรียนออนไลน์:

- Python Tuples - W3Schools
- Python Sets - GeeksforGeeks
- Python Dictionaries - Real Python

คอร์สเรียน:

- Coursera: Data Collection and Processing with Python โดย University of Michigan
- Udemy: Complete Python Bootcamp: Go from zero to hero in Python 3

ภาคผนวก A7.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). เอกสาร Python - โครงสร้างข้อมูล. จาก https://docs.python.org/3/tutorial/datastructures.html
- Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook* (ฉบับที่ 3). สำนักพิมพ์ O'Reilly Media.

ภาคผนวก A7.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การใช้งาน Tuple เบื้องต้น

- เขียนโปรแกรม Python เพื่อสร้าง Tuple และแสดงสมาชิกทั้งหมด
- สร้าง Tuple ที่มีตัวเลข และหาค่ามากสุด/น้อยสุดจาก Tuple

แบบฝึกหัดที่ 2: การใช้งาน Set เบื้องต้น

- เขียนโปรแกรมสร้าง Set จากลิสต์ที่มีข้อมูลซ้ำกัน
- สร้างเซต 2 ชุดและดำเนินการ union, intersection และ difference

แบบฝึกหัดที่ 3: การใช้งาน Dictionary เบื้องต้น

- เขียนโปรแกรมสร้าง Dictionary และแสดงคีย์และค่าทั้งหมด
- สร้าง Dictionary ของนักเรียนและเกรด แล้วแสดงนักเรียนที่ได้คะแนนสูงสุด

แบบฝึกหัดที่ 4: การใช้งานขั้นสูงกับ Tuple, Set และ Dictionary

- เขียนฟังก์ชันที่รับลิสต์ของ Tuple (ชื่อ, คะแนน) แล้วแปลงเป็น Dictionary
- เขียนโปรแกรมที่รับข้อความประโยคหนึ่ง แล้วคืนค่าความถี่ของแต่ละคำในรูปแบบ Dictionary

ภาคผนวก A7.4 ตัวอย่างการใช้งานจริง

```python
#         Tuple
fruits = ("apple", "banana", "cherry")
for fruit in fruits:
    print(fruit)

#           Tuple
numbers = (10, 20, 30, 40, 50)
print("Max:", max(numbers))
print("Min:", min(numbers))
```

Listing 7.22: การสร้างและแสดง Tuple

```python
#           Set
numbers = [1, 2, 2, 3, 4, 4, 5]
unique_numbers = set(numbers)
print(unique_numbers)

#       Set: union, intersection, difference
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print("Union:", set1.union(set2))
print("Intersection:", set1.intersection(set2))
print("Difference:", set1.difference(set2))
```

Listing 7.23: การลบค่าซ้ำในลิสต์โดยใช้ Set

```python
#        Dictionary
student_grades = {"Alice": 85, "Bob": 92, "Charlie": 78}
for student, grade in student_grades.items():
    print(f"{student}: {grade}")

#
highest_grade = max(student_grades.values())
top_student = [name for name, grade in student_grades.items() if grade ==
    highest_grade]
print("Top student(s):", top_student)
```

Listing 7.24: การสร้างและใช้งาน Dictionary

```python
#         Tuple    Dictionary
def tuples_to_dict(tuples_list):
    return dict(tuples_list)

students = [("Alice", 85), ("Bob", 92), ("Charlie", 78)]
student_dict = tuples_to_dict(students)
print(student_dict)

#
def word_frequency(sentence):
    words = sentence.split()
    freq_dict = {}
    for word in words:
        if word in freq_dict:
            freq_dict[word] += 1
        else:
            freq_dict[word] = 1
    return freq_dict

sentence = "the quick brown fox jumps over the lazy dog the quick brown fox
    "
print(word_frequency(sentence))
```

Listing 7.25: ฟังก์ชันแปลงลิสต์ของ Tuple เป็น Dictionary และนับความถี่คำ

บทที่ 8

การจัดการไฟล์และการดำเนินการ I/O

บทนี้นำเสนอภาพรวมอย่างครบถ้วนเกี่ยวกับการจัดการไฟล์และการดำเนินการ I/O ในภาษา Python โดยครอบคลุมเทคนิคสำคัญสำหรับการจัดการไฟล์ในโปรแกรมอย่างมีประสิทธิภาพ เริ่มต้นด้วยพื้นฐานของการเปิดไฟล์ด้วยโหมดต่าง ๆ (`'r'`, `'w'`, `'a'`, `'b'`, `'t'` และ `'+'`) ซึ่งแต่ละโหมดมีวัตถุประสงค์เฉพาะสำหรับการโต้ตอบกับไฟล์ เน้นความสำคัญของการปิดไฟล์ด้วยเมธอด `close()` เพื่อให้แน่ใจว่าทรัพยากรระบบได้รับการปล่อยอย่างเหมาะสม โดยมีการแนะนำการใช้คำสั่ง `with` เป็นแนวปฏิบัติที่ดีในการจัดการบริบทของไฟล์ ซึ่งจะจัดการการปิดไฟล์ให้อัตโนมัติแม้ในกรณีที่เกิดข้อยกเว้น

เนื้อหาจะพัฒนาต่อไปสู่รายละเอียดของวิธีการอ่านไฟล์ เช่น การอ่านไฟล์ทั้งหมดด้วย `read()` การอ่านทีละบรรทัดด้วย `readline()` และการอ่านทุกบรรทัดด้วย `readlines()` การเขียนไฟล์ครอบคลุมถึงการใช้เมธอด `write()` ซึ่งแสดงวิธีสร้างและแก้ไขเนื้อหาไฟล์ การเพิ่มข้อมูลลงในไฟล์โดยไม่เขียนทับข้อมูลเดิมจะอธิบายผ่านโหมด `append()`

นอกจากนี้ยังสำรวจการดำเนินการรับและแสดงผลด้วยตัวอย่างจริง โดยแสดงการใช้ `input()` สำหรับรับค่าจากผู้ใช้ และ `print()` สำหรับแสดงผลที่หน้าจอ บทนี้สรุปด้วยตัวอย่างเชิงปฏิบัติที่แสดงการใช้งานจริง เช่น การคัดลอกเนื้อหาไฟล์ การนับจำนวนคำในไฟล์ และการเขียนข้อมูลจากผู้ใช้ลงในไฟล์ ซึ่งตัวอย่างเหล่านี้ช่วยเสริมความเข้าใจและแสดงให้เห็นถึงความยืดหยุ่นของการจัดการไฟล์ใน Python ที่ผู้อ่านสามารถนำไปประยุกต์ใช้ในโปรแกรมของตนได้อย่างมั่นใจ

8.1 การจัดการไฟล์

การจัดการไฟล์เป็นส่วนสำคัญของการเขียนโปรแกรมที่เกี่ยวข้องกับการสร้าง อ่าน เขียน และปิดไฟล์ ซึ่งช่วยให้โปรแกรมสามารถเก็บข้อมูลไว้ระหว่างการทำงานหลายครั้ง Python มีฟังก์ชันและเมธอดในตัวที่ช่วยในการดำเนินการเหล่านี้ได้อย่างมีประสิทธิภาพ ทำให้สามารถโต้ตอบกับไฟล์ได้อย่างราบรื่น ฟังก์ชันเหล่านี้ช่วยให้นักพัฒนาสามารถเปิดไฟล์ในโหมดต่าง ๆ (อ่าน เขียน เพิ่มข้อมูล) อ่านและเขียนไฟล์ และปิดไฟล์เพื่อปล่อยทรัพยากรระบบ การเรียนรู้การจัดการไฟล์จึงเป็นทักษะพื้นฐานที่จำเป็นตั้งแต่การบันทึกข้อมูลอย่างง่าย ไปจนถึงการประมวลผลข้อมูลที่ซับซ้อน

แนวคิดสำคัญเกี่ยวกับไฟล์:

- **การเปิดไฟล์**: `open(filename, mode)`
- **โหมดของไฟล์**:
  - `'r'`: อ่าน
  - `'w'`: เขียน
  - `'a'`: เพิ่มข้อมูล
  - `'b'`: โหมดไบนารี
  - `'t'`: โหมดข้อความ
  - `'+'`: อ่าน/เขียน
- **การปิดไฟล์**: `file.close()`
- **การใช้ with statement**: ปิดไฟล์ให้อัตโนมัติ
- **การอ่านไฟล์**:
  - `read()`: อ่านทั้งไฟล์
  - `readline()`: อ่านทีละบรรทัด
  - `readlines()`: อ่านทุกบรรทัด
- **การเขียนไฟล์**: `file.write()`
- **การเพิ่มข้อมูล**: ใช้โหมด `'a'`

8.1.1 การเปิดไฟล์

การเปิดไฟล์ใน Python ใช้ฟังก์ชัน `open()` ซึ่งต้องการอาร์กิวเมนต์สองตัว ได้แก่ ชื่อไฟล์และโหมดการเปิด โหมดนี้จะกำหนดลักษณะการใช้งานไฟล์ เช่น อ่าน เขียน หรือเพิ่มข้อมูล ฟังก์ชัน `open()` จะส่งคืนอ็อบเจกต์ไฟล์ ซึ่งสามารถใช้ดำเนินการต่าง ๆ กับไฟล์ได้ การเปิดไฟล์อย่างถูกต้องเป็นขั้นตอนแรกของการจัดการไฟล์ เพื่อให้สามารถทำงานกับไฟล์ได้ตามต้องการ

ไวยากรณ์:

```
file_object = open(file_name, mode)
```

โหมดต่าง ๆ:

- `'r'`: โหมดอ่าน (ค่าเริ่มต้น) เปิดไฟล์เพื่ออ่าน
- `'w'`: โหมดเขียน เปิดไฟล์เพื่อเขียน (สร้างไฟล์ใหม่หรือล้างข้อมูลไฟล์เดิม)
- `'a'`: โหมดเพิ่มข้อมูล เปิดไฟล์เพื่อเพิ่มข้อมูล (สร้างไฟล์ใหม่หากยังไม่มี)
- `'b'`: โหมดไบนารี เปิดไฟล์ในรูปแบบไบนารี

- `'t'`: โหมดข้อความ (ค่าเริ่มต้น) เปิดไฟล์ในรูปแบบข้อความ
- `'+'`: โหมดอัปเดต เปิดไฟล์เพื่ออ่านและเขียน

ตัวอย่าง:

```
# Opening a file in read mode
file = open("example.txt", "r")
```

Listing 8.1: Example of Opening File

8.1.2 การปิดไฟล์

การปิดไฟล์ใน Python เป็นสิ่งสำคัญเพื่อให้มั่นใจว่าทรัพยากรที่เกี่ยวข้องกับไฟล์ได้รับการปล่อยออกอย่างถูกต้อง และข้อมูลที่เขียนลงไปในไฟล์ถูกบันทึกอย่างสมบูรณ์ ซึ่งทำได้โดยการใช้เมธอด `close()` กับอ็อบเจกต์ไฟล์ หากไม่ปิดไฟล์อาจทำให้เกิดการใช้หน่วยความจำมากเกินไปหรือเกิดการเสียหายของข้อมูล เมื่อไฟล์ถูกปิด จะไม่สามารถใช้งานได้อีกต่อไป และหากดำเนินการใด ๆ ต่อไปจะเกิดข้อผิดพลาด ดังนั้นควรปิดไฟล์ทุกครั้งหลังใช้งานเสร็จเพื่อรักษาประสิทธิภาพของโปรแกรมและความถูกต้องของข้อมูล

ตัวอย่าง:

```
# Closing the file
file.close()
```

Listing 8.2: Example of Closing File

8.1.3 การใช้ with statement

คำสั่ง `with` ใช้เพื่อให้มั่นใจว่าไฟล์จะถูกปิดอย่างถูกต้องหลังจากโค้ดในบล็อกเสร็จสิ้น แม้จะเกิดข้อยกเว้นก็ตาม การใช้ `with` ช่วยจัดการทรัพยากรไฟล์ได้อย่างมีประสิทธิภาพ เพราะมันจะปิดไฟล์ให้อัตโนมัติเมื่อออกจากบล็อกของโค้ด ทำให้ไม่จำเป็นต้องเรียก `close()` โดยตรง และช่วยให้โค้ดดูสะอาดและอ่านง่ายขึ้น

ตัวอย่าง:

```
# Using with statement to open and close a file
with open("example.txt", "r") as file:
    contents = file.read()
    print(contents)
```

Listing 8.3: Using with statement to open and close a file

8.2 การเขียนไฟล์

การเขียนไฟล์ในภาษา Python ใช้เมธอด `write()` ซึ่งช่วยให้สามารถส่งออกข้อความ (string) ลงในไฟล์ได้ เมธอดนี้มีประโยชน์สำหรับการสร้างไฟล์ใหม่หรือเขียนทับไฟล์ที่มีอยู่ด้วยข้อมูลใหม่ เมื่อเปิดไฟล์ในโหมดเขียน (`'w'`), Python จะสร้างไฟล์ใหม่หากไฟล์ยังไม่มีอยู่ หรือจะล้างเนื้อหาเดิมหากไฟล์มีอยู่แล้ว การใช้คำสั่ง `with` ขณะเขียน

ไฟล์เป็นแนวทางที่สำคัญ เนื่องจากจะช่วยให้ไฟล์ถูกปิดอย่างถูกต้องหลังจากดำเนินการเขียนเสร็จสิ้น ซึ่งช่วยป้องกันการสูญหายของข้อมูลและรับประกันว่าการเปลี่ยนแปลงทั้งหมดได้รับการบันทึกอย่างถูกต้อง

8.2.1 การเขียนลงในไฟล์

ในภาษา Python เมธอด `write()` ใช้สำหรับเขียนข้อความลงในไฟล์ สามารถใช้เพื่อเพิ่มเนื้อหาลงในไฟล์ ไม่ว่าจะเป็นการสร้างไฟล์ใหม่หรือแก้ไขไฟล์ที่มีอยู่แล้ว เมื่อเรียกใช้ `write()` ข้อความที่กำหนดจะถูกเขียนในตำแหน่งปัจจุบันของไฟล์ หากเปิดไฟล์ในโหมดเขียน (`'w'`), ไฟล์จะถูกล้างข้อมูลเดิมออกก่อน เมธอดนี้มีความสำคัญในการบันทึกข้อมูล การสร้างรายงาน และการจัดเก็บข้อมูลจากผู้ใช้

```python
# Writing to a file
with open("example.txt", "w") as file:
    file.write("Hello, World!\n")
    file.write("This is a new line.\n")
```

Listing 8.4: Writing to a File

8.2.2 การเพิ่มข้อมูลลงในไฟล์

เมธอด `append()` ใช้สำหรับเพิ่มเนื้อหาลงท้ายไฟล์โดยไม่ลบข้อมูลที่มีอยู่เดิม เมื่อเปิดไฟล์ในโหมดเพิ่มข้อมูล (`'a'`), ข้อมูลใหม่จะถูกเขียนต่อท้ายจากตำแหน่งสุดท้ายของไฟล์ โดยที่ข้อมูลเก่าจะยังคงอยู่ เมธอดนี้มีประโยชน์สำหรับการบันทึกข้อมูล (logging) การเก็บข้อมูลสถิติ หรือการเพิ่มข้อมูลทีละส่วนในระยะยาว การใช้คำสั่ง `with` จะช่วยให้แน่ใจว่าไฟล์ถูกปิดอย่างถูกต้องหลังจากเพิ่มข้อมูล ซึ่งช่วยให้การจัดการทรัพยากรมีประสิทธิภาพและข้อมูลมีความถูกต้องครบถ้วน

```python
# Appending to a file
with open("example.txt", "a") as file:
    file.write("This line is appended.\n")
```

Listing 8.5: Appending to a File

8.3 การอ่านไฟล์

การอ่านไฟล์ในภาษา Python สามารถทำได้โดยใช้หลายเมธอดที่มีอยู่ในอ็อบเจกต์ไฟล์ ได้แก่ `read()` สำหรับอ่านเนื้อหาทั้งหมดของไฟล์เป็นสตริงเดียว `readline()` สำหรับอ่านทีละบรรทัด และ `readlines()` สำหรับอ่านทุกบรรทัดเป็นลิสต์ เมธอดเหล่านี้ให้ความยืดหยุ่นในการจัดการเนื้อหาไฟล์ ทำให้สามารถประมวลผลไฟล์ขนาดใหญ่ได้อย่างมีประสิทธิภาพ หรือเลือกอ่านเฉพาะส่วนที่ต้องการ การเข้าใจวิธีการเหล่านี้เป็นสิ่งสำคัญสำหรับการจัดการไฟล์และประมวลผลข้อมูลอย่างมีประสิทธิภาพใน Python

8.3.1 การอ่านไฟล์ทั้งหมด

การอ่านไฟล์ทั้งหมดในภาษา Python ใช้เมธอด `read()` ซึ่งจะอ่านเนื้อหาทั้งหมดของไฟล์และส่งคืนเป็นสตริงเดียว เหมาะสำหรับกรณีที่ต้องการประมวลผลหรือวิเคราะห์ข้อมูลทั้งหมดในคราวเดียว อย่างไรก็ตามควรระวังในกรณีที่ไฟล์มีขนาดใหญ่ เพราะอาจใช้หน่วยความจำมาก การใช้คำสั่ง `with` เพื่อเปิดไฟล์จะช่วยให้ไฟล์ถูกปิดอย่างถูกต้องหลังจากอ่านเสร็จ ซึ่งช่วยจัดการทรัพยากรได้อย่างเหมาะสม

```python
with open("example.txt", "r") as file:
    contents = file.read()
    print(contents)
```

Listing 8.6: Reading Entire File

8.3.2 การอ่านแต่ละบรรทัด

การอ่านไฟล์ทีละบรรทัดใน Python สามารถทำได้โดยใช้เมธอด `readline()` และ `readlines()` โดย `readline()` จะอ่านข้อมูลทีละบรรทัด เหมาะกับการจัดการไฟล์ขนาดใหญ่โดยไม่ใช้หน่วยความจำมาก ส่วน `readlines()` จะอ่านทุกบรรทัดและส่งคืนเป็นลิสต์ เหมาะกับไฟล์ขนาดเล็ก หรือเมื่อต้องการจัดการข้อมูลทั้งไฟล์พร้อมกัน การใช้ทั้งสองเมธอดร่วมกับคำสั่ง `with` ช่วยให้แน่ใจว่าไฟล์ถูกปิดหลังการอ่าน ช่วยให้โค้ดสะอาดและมีประสิทธิภาพในการใช้ทรัพยากร

```python
with open("example.txt", "r") as file:
    line = file.readline()
    while line:
        print(line.strip())
        line = file.readline()
```

Listing 8.7: Reading one line at a time

```python
with open("example.txt", "r") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())
```

Listing 8.8: Reading all lines into a list

8.3.3 ตัวอย่างการประมวลผลไฟล์ข้อความ

ตัวอย่าง: ใช้ลูปเขียนข้อมูลลงไฟล์:

```python
# This program prompts the user for sales amounts
# and writes those amounts to the sales.txt file.
# Get the number of days.
num_days = int(input('For how many days do you have sales? '))
# Open a new file named sales.txt using with statement.
with open('sales.txt', 'w') as sales_file:
    # Get the amount of sales for each day and write it to the file.
    for count in range(1, num_days + 1):
        # Get the sales for a day.
        sales = float(input(f'Enter the sales for day #{count}: '))
        # Write the sales amount to the file.
        sales_file.write(str(sales) + '\n')

print('Data written to sales.txt.')
```

Listing 8.9: Loop Writing File

ตัวอย่าง: ใช้ For Loop อ่านไฟล์

```
# This program uses the for loop to read
# all of the values in the sales.txt file.
# Open the sales.txt file for reading using with statement.
with open('sales.txt', 'r') as sales_file:
    # Read all the lines from the file.
    for line in sales_file:
        # Convert line to a float.
        amount = float(line)
        # Format and display the amount.
        print(format(amount, '.2f'))
```

Listing 8.10: For Loop Reading File

ตัวอย่าง: ใช้ While Loop อ่านไฟล์

```
# This program reads all of the values in the sales.txt file.
# Open the sales.txt file for reading using with statement.
with open('sales.txt', 'r') as sales_file:
    # Read the first line from the file, but don't convert to a number yet.
    # We still need to test for an empty string.
    line = sales_file.readline()
    # Read until the empty string is return.
    while line != '':
        # Convert line to a float.
        amount = float(line)
        # Format and display the amount.
        print(format(amount, '.2f'))
        # Read the next line.
        line = sales_file.readline()
```

Listing 8.11: Using While Loop Reading File

ตัวอย่าง: การทำงานกับข้อมูลพนักงานในรูปแบบข้อความ

```
# Get the number of employee records to create.
num_emps = int(input('How many employee records do you want to create? '))
# Open a file for writing using with statement.
with open('employees.txt', 'w') as emp_file:
    # Get each employee's data and write it to the file.
    for count in range(1, num_emps + 1):
        # Get the data for an employee.
        print('Enter data for employee #', count, sep='')
        name = input('Name: ')
        id_num = input('ID number: ')
        dept = input('Department: ')
        # Write the data as a record to the file.
        emp_file.write(name + '\n')
        emp_file.write(id_num + '\n')
        emp_file.write(dept + '\n')
        # Display a blank line.
        print()

print('Employee records written to employees.txt.')
```

Listing 8.12: Working with Text Record

8.4 การทำงานกับเรคอร์ดและไฟล์ไบนารี

การทำงานกับเรคอร์ดและไฟล์ไบนารีในภาษา Python ช่วยให้สามารถจัดการข้อมูลได้อย่างมีประสิทธิภาพยิ่งขึ้น โดยเฉพาะเมื่อเกี่ยวข้องกับข้อมูลที่มีโครงสร้างหรือไม่ใช่ข้อความ เรคอร์ดมักจัดเก็บเป็นชุดของฟิลด์ และไฟล์ไบนารีสามารถจัดเก็บเรคอร์ดเหล่านี้ในรูปแบบที่กระชับและมีประสิทธิภาพ ไฟล์ไบนารีจะแตกต่างจากไฟล์ข้อความตรงที่เก็บข้อมูลในรูปแบบไบนารีดิบ ซึ่งมีความเร็วและประหยัดพื้นที่มากกว่าเมื่อใช้กับข้อมูลบางประเภท

8.4.1 การเขียนเรคอร์ดลงในไฟล์ไบนารี

ในการเขียนเรคอร์ดลงในไฟล์ไบนารี ต้องแปลงข้อมูลให้อยู่ในรูปแบบไบนารีก่อน โดยโมดูล `struct` ในภาษา Python มักถูกใช้เพื่อวัตถุประสงค์นี้ ซึ่งให้ฟังก์ชันในการ pack ข้อมูลเป็นไบนารี และ unpack กลับสู่รูปแบบเดิมได้

```
import struct

# Define a record as a tuple
record = (1, 'John Doe', 20, 3.75)
# Open a binary file for writing
with open("records.bin", "wb") as file:
    # Pack the record into binary format
    data = struct.pack('i20sif', record[0], record[1].encode(),
           record[2], record[3])
    # Write the packed data to the file
    file.write(data)
```

Listing 8.13: Writing a Record to a Binary File

ในตัวอย่างนี้ เรคอร์ดคือ tuple ที่ประกอบด้วยจำนวนเต็ม (ID), สตริง (ชื่อ), จำนวนเต็ม (อายุ) และค่าทศนิยม (GPA) ฟังก์ชัน `struct.pack()` จะเปลี่ยน tuple นี้ให้อยู่ในรูปแบบไบนารี แล้วเขียนลงไฟล์

คำอธิบายรูปแบบ `"i20sif"`

สตริงรูปแบบ `"i20sif"` ที่ใช้ในฟังก์ชัน `struct.pack()` ระบุโครงสร้างของข้อมูล:

- `i`: แทนจำนวนเต็ม 4 ไบต์ (32 บิต) ใช้สำหรับฟิลด์ ID และ Age
- `20s`: แทนสตริงขนาด 20 ไบต์ อักษร `s` หมายถึงสตริง และตัวเลข `20` บ่งชี้ว่าจะใช้พื้นที่ 20 ไบต์ในไฟล์ไบนารี ถ้าสตริงสั้นกว่าจะถูกเติมด้วย null byte (`\x00`) ถ้ายาวกว่าจะถูกตัด ใช้กับฟิลด์ Name
- `f`: แทนจำนวนทศนิยมขนาด 4 ไบต์ (32 บิต) ใช้กับฟิลด์ GPA

ดังนั้น `"i20sif"` จึงหมายถึงโครงสร้างข้อมูลที่ประกอบด้วยจำนวนเต็ม 4 ไบต์, สตริง 20 ไบต์, จำนวนเต็ม 4 ไบต์ และทศนิยม 4 ไบต์ ซึ่งช่วยให้ข้อมูลถูกจัดเก็บอย่างเป็นระเบียบในไฟล์ไบนารี

```python
import struct

# Get the number of records to write
num_records = int(input("How many records do you want to create? "))
# Open a binary file for writing
with open("records.bin", "wb") as file:
    # Loop to get each record's data
    for _ in range(num_records):
        # Get data from the user
        id_num = int(input("Enter ID: "))
        name = input("Enter Name: ")
        age = int(input("Enter Age: "))
        gpa = float(input("Enter GPA: "))
        # Pack the record into binary format
        data = struct.pack('i20sif', id_num, name.encode(), age, gpa)
        # Write the packed data to the file
        file.write(data)

print(f"{num_records} records have been written to records.bin")
```

Listing 8.14: Writing Multiple Records to a Binary File

8.4.2 การอ่านเรคอร์ดจากไฟล์ไบนารี

ในการอ่านเรคอร์ดจากไฟล์ไบนารี จะใช้ฟังก์ชัน `struct.unpack()` เพื่อแปลงข้อมูลไบนารีกลับมาเป็นรูปแบบเดิม

```python
import struct

# Open the binary file for reading
with open("records.bin", "rb") as file:
    # Read the binary data from the file
    data = file.read(struct.calcsize('i20sif'))
    # Unpack the data back into a tuple
    record = struct.unpack('i20sif', data)
    # Decode the string field and remove trailing null bytes
```

```
    record = (record[0], record[1].decode().strip('\x00'), record[2],
        record[3])
    print(f"ID: {record[0]}, Name: {record[1]}, Age: {record[2]}, GPA: {
        record[3]}")
```

Listing 8.15: Reading a Record from a Binary File

ในตัวอย่างนี้ จะอ่านข้อมูลไบนารีจากไฟล์ แล้วใช้ `struct.unpack()` แปลงกลับเป็น tuple จากนั้นแปลงสตริงจาก byte เป็นข้อความ และลบ null byte ที่ตามท้ายออก

8.4.3 การทำงานกับหลายเรคอร์ด

เมื่อทำงานกับหลายเรคอร์ด มักจะใช้ลูปเพื่ออ่านและแปลงแต่ละเรคอร์ดจากไฟล์ทีละรายการ

```
import struct

# Open the binary file for reading
with open("records.bin", "rb") as file:
    record_size = struct.calcsize('i20sif')
    while True:
        data = file.read(record_size)
        if not data:
            break
        record = struct.unpack('i20sif', data)
        record = (record[0], record[1].decode().strip('\x00'), record[2],
            record[3])
        print(f"ID: {record[0]}, Name: {record[1]}, Age: {record[2]}, GPA:
            {record[3]}")
```

Listing 8.16: Reading Multiple Records

ตัวอย่างนี้แสดงการอ่านหลายเรคอร์ดจากไฟล์ไบนารีภายในลูป โดยแปลงแต่ละเรคอร์ดกลับและประมวลผลทีละรายการ

ในการอ่านเรคอร์ดที่สองจากไฟล์ไบนารี ต้องข้ามเรคอร์ดแรกก่อน จากนั้นจึงอ่านเรคอร์ดที่สอง ดังตัวอย่างต่อไปนี้:

```
import struct

# Define the format string
record_format = 'i20sif'
record_size = struct.calcsize(record_format)
# Open the binary file for reading
with open("records.bin", "rb") as file:
    # Skip the first record
    file.seek(record_size)
    # Read the second record
    data = file.read(record_size)
    # Unpack the data back into a tuple
    record = struct.unpack(record_format, data)
    # Decode the string field and remove trailing null bytes
    record = (record[0], record[1].decode().strip('\x00'), record[2],
        record[3])

    print(f"ID: {record[0]}, Name: {record[1]}, Age: {record[2]}, GPA: {
        record[3]}")
```

Listing 8.17: Reading the second record from a binary file

คำอธิบาย:

- `record_format = 'i20sif'`: เป็นสตริงรูปแบบที่กำหนดโครงสร้างของแต่ละเรคอร์ดในไฟล์
- `record_size = struct.calcsize(record_format)`: ใช้คำนวณขนาดของเรคอร์ดแต่ละรายการในหน่วยไบต์
- `file.seek(record_size)`: ข้ามเรคอร์ดแรกโดยเลื่อนตำแหน่ง pointer ไปยังจุดเริ่มต้นของเรคอร์ดที่สอง
- `data = file.read(record_size)`: อ่านเรคอร์ดที่สองจากไฟล์
- `struct.unpack(record_format, data)`: แปลงข้อมูลไบนารีกลับเป็นโครงสร้างเดิม
- `record[1].decode().strip('\x00')`: แปลงสตริงจาก byte เป็นข้อความ และลบ null byte (\x00) ที่ตามท้าย

โค้ดนี้จะอ่านและแสดงข้อมูลของเรคอร์ดที่สองจากไฟล์ไบนารีได้อย่างถูกต้อง

8.5 การใช้โหมด + ใน Python

โหมดไฟล์ + ในภาษา Python ช่วยให้สามารถเปิดไฟล์เพื่อทั้งการอ่านและเขียนได้ โดยพฤติกรรมจะขึ้นอยู่กับโหมดที่ใช้งานร่วมกัน (`r+`, `w+`, หรือ `a+`) ดังนี้:

- `r+`: เปิดไฟล์เพื่ออ่านและเขียน ไฟล์จะต้องมีอยู่แล้ว มิฉะนั้นจะเกิดข้อผิดพลาด
- `w+`: เปิดไฟล์เพื่ออ่านและเขียน หากไฟล์มีอยู่แล้วจะลบเนื้อหาเก่าออก หากยังไม่มีจะสร้างไฟล์ใหม่
- `a+`: เปิดไฟล์เพื่ออ่านและเขียน ตัวชี้ตำแหน่งไฟล์จะอยู่ที่ท้ายไฟล์ (หากมีอยู่) การเขียนจะถูกเพิ่มต่อท้ายเสมอ หากไฟล์ยังไม่มีจะสร้างไฟล์ใหม่

8.5.1 ตัวอย่างการใช้โหมด `r+`

ตัวอย่างต่อไปนี้แสดงการใช้โหมด `r+` เพื่ออ่านและเพิ่มเนื้อหาลงในไฟล์ที่มีอยู่แล้ว:

```
# Example using r+ mode (read and write)
def example_r_plus_mode():
    try:
        # Open the file for reading and writing
        with open('example.txt', 'r+') as file:
            # Read the current contents of the file
            content = file.read()
            print("Current content of the file:")
            print(content)

            # Move the file pointer back to the beginning
            file.seek(0, 0)

            # Write new content to the file
            file.write("New content added.\n")

            # Move the file pointer to the end of the file
            file.seek(0, 2)

```

```
            # Append more content at the end
            file.write("Appending more content at the end.\n")

            # Read the file again after modification
            file.seek(0)
            updated_content = file.read()
            print("\nUpdated content of the file:")
            print(updated_content)

    except FileNotFoundError:
        print("The file does not exist.")

example_r_plus_mode()
```

Listing 8.18: Example using r+ mode (read and write)

คำอธิบาย:

- `with open('example.txt', 'r+') as file:`: เปิดไฟล์ `example.txt` เพื่ออ่านและเขียน ไฟล์ต้องมีอยู่แล้ว
- `file.read()`: อ่านเนื้อหาปัจจุบันของไฟล์
- `file.seek(0, 0)`: ย้ายตำแหน่งตัวชี้ไฟล์กลับไปจุดเริ่มต้น
- `file.write("New content added.\n")`: เขียนเนื้อหาใหม่ที่จุดเริ่มต้นของไฟล์
- `file.seek(0, 2)`: ย้ายตัวชี้ไฟล์ไปยังจุดสิ้นสุด
- `file.write("Appending more content at the end.\n")`: เขียนเนื้อหาต่อท้ายไฟล์
- `file.seek(0)`: ย้ายตัวชี้ไฟล์กลับไปจุดเริ่มต้นอีกครั้งเพื่ออ่านเนื้อหาที่อัปเดตแล้ว

8.5.2 ตัวอย่างการใช้โหมด w+

ตัวอย่างต่อไปนี้แสดงการใช้โหมด `w+` เพื่อสร้างไฟล์ใหม่ เขียน และอ่านจากไฟล์:

```
# Example using w+ mode (write and read)
def example_w_plus_mode():
    # Open the file for reading and writing (truncating it)
    with open('example_w+.txt', 'w+') as file:
        # Write some content to the file
        file.write("This is the first line in the file.\n")
        file.write("This is the second line in the file.\n")

        # Move the file pointer back to the beginning
        file.seek(0)

        # Read the content of the file
        content = file.read()
        print("Content of the file:")
        print(content)

example_w_plus_mode()
```

Listing 8.19: Example using w+ mode (write and read)

คำอธิบาย:

- `with open('example_w+.txt', 'w+') as file:`: เปิดไฟล์ `example_w+.txt` เพื่ออ่านและเขียน หากไฟล์มีอยู่จะลบเนื้อหาเดิม หากไม่มีจะสร้างใหม่
- `file.write(...)`: เขียนเนื้อหาลงในไฟล์
- `file.seek(0)`: ย้ายตัวชี้ไฟล์กลับไปจุดเริ่มต้น
- `file.read()`: อ่านเนื้อหาหลังจากเขียนลงไป

8.5.3 ตัวอย่างการใช้โหมด a+

ตัวอย่างต่อไปนี้แสดงการใช้โหมด `a+` เพื่อเปิดไฟล์ เพิ่มข้อมูล และอ่านเนื้อหาจากไฟล์:

```
# Example using a+ mode (append and read)
def example_a_plus_mode():
    # Open the file for reading and writing (appending)
    with open('example_a+.txt', 'a+') as file:
        # Move the file pointer to the beginning to read the content
        file.seek(0)

        # Read the current content of the file
        content = file.read()
        print("Current content of the file:")
        print(content)

        # Append new content to the file
        file.write("Appending a new line at the end.\n")

        # Move the file pointer to the beginning again to read the updated
            content
        file.seek(0)
        updated_content = file.read()
        print("\nUpdated content of the file:")
        print(updated_content)

example_a_plus_mode()
```

Listing 8.20: Example using a+ mode (append and read)

คำอธิบาย:

- `with open('example_a+.txt', 'a+') as file:`: เปิดไฟล์ `example_a+.txt` เพื่ออ่านและเขียน โดยตัวชี้จะเริ่มที่ท้ายไฟล์ ถ้าไฟล์ยังไม่มีจะสร้างขึ้นใหม่
- `file.seek(0)`: ย้ายตัวชี้ไฟล์ไปจุดเริ่มต้นเพื่ออ่านเนื้อหา
- `file.write(...)`: เพิ่มข้อมูลต่อท้ายไฟล์
- `file.seek(0)`: ย้ายตัวชี้ไฟล์กลับไปจุดเริ่มต้นเพื่ออ่านเนื้อหาที่อัปเดต

ตัวอย่างเหล่านี้แสดงให้เห็นว่าโหมด `+` สามารถใช้ร่วมกับ `r`, `w` หรือ `a` เพื่อเปิดไฟล์สำหรับทั้งการอ่านและเขียนได้อย่างยืดหยุ่น

8.5.4 ข้อควรพิจารณาในการใช้งานไฟล์ไบนารี

- **Endianness (ลำดับไบต์)**: ควรระวังลำดับไบต์ของระบบ (endianness) เมื่อทำงานกับไฟล์ไบนารี โดยเฉพาะกรณีที่ไฟล์จะถูกอ่านในแพลตฟอร์มที่แตกต่างกัน
- **ขนาดไฟล์**: ไฟล์ไบนารีมักมีขนาดกะทัดรัดกว่าไฟล์ข้อความ ซึ่งช่วยประหยัดพื้นที่ดิสก์และเพิ่มประสิทธิภาพการทำงาน
- **ความถูกต้องของข้อมูล**: ต้องแน่ใจว่าชนิดข้อมูลและโครงสร้างที่ใช้ในการ pack และ unpack ตรงกันอย่างถูกต้อง มิฉะนั้นข้อมูลอาจเสียหายหรือไม่สามารถอ่านได้

โดยการเข้าใจวิธีการทำงานกับเรคอร์ดและไฟล์ไบนารี คุณจะสามารถจัดการข้อมูลที่มีโครงสร้างในภาษา Python ได้อย่างมีประสิทธิภาพ เช่น ข้อมูลในฐานข้อมูล ภาพ และชนิดข้อมูลที่ซับซ้อนอื่น ๆ

8.6 ตัวอย่างการใช้งานจริง

ตัวอย่างการใช้งานจริงในบทนี้คือการคัดลอกเนื้อหาจากไฟล์หนึ่งไปยังอีกไฟล์หนึ่ง กระบวนการนี้ใช้ฟังก์ชัน `open()` เพื่ออ่านข้อมูลจากไฟล์ต้นทาง และเขียนไปยังไฟล์ปลายทาง การใช้คำสั่ง `with` ช่วยให้มั่นใจได้ว่าไฟล์ทั้งสองจะถูกปิดอย่างถูกต้องหลังจากการดำเนินการ เสริมสร้างความปลอดภัยในการใช้งานทรัพยากร นอกจากนี้ตัวอย่างนี้ยังแสดงให้เห็นถึงการใช้เมธอด `read()` เพื่อดึงข้อมูล และเมธอด `write()` เพื่อเขียนข้อมูล ตัวอย่างนี้เน้นให้เห็นถึงความสำคัญของเทคนิคการจัดการไฟล์ในการจัดการและประมวลผลข้อมูลภายในโปรแกรมอย่างมีประสิทธิภาพ ซึ่งสอดคล้องกับแนวคิดหลักของบทนี้

8.6.1 ตัวอย่าง: คัดลอกเนื้อหาจากไฟล์หนึ่งไปยังอีกไฟล์

การใช้การจัดการไฟล์เพื่อคัดลอกเนื้อหาจากไฟล์หนึ่งไปยังอีกไฟล์ เป็นกระบวนการพื้นฐานที่แสดงการประยุกต์ใช้งานจริงของการอ่านและเขียนไฟล์ โดยทั่วไปจะเริ่มจากการเปิดไฟล์ต้นทางในโหมดอ่าน ใช้เมธอดอย่าง `read()` เพื่ออ่านเนื้อหา แล้วเปิดไฟล์ปลายทางในโหมดเขียนเพื่อบันทึกข้อมูล การใช้คำสั่ง `with` ช่วยให้มั่นใจได้ว่าไฟล์ทั้งสองถูกปิดอย่างถูกต้อง ป้องกันการสูญเสียทรัพยากร และรักษาความสมบูรณ์ของข้อมูล ตัวอย่างนี้แสดงให้เห็นว่าเทคนิคการจัดการไฟล์สามารถนำมาใช้จัดการข้อมูลในโปรแกรมได้อย่างมีประสิทธิภาพ ซึ่งเป็นทักษะสำคัญสำหรับนักเขียนโปรแกรม

ตัวอย่าง:

```
# Copying content from source.txt to destination.txt
with open("source.txt", "r") as source_file:
    content = source_file.read()
with open("destination.txt", "w") as destination_file:
    destination_file.write(content)
```

Listing 8.21: Copying Content from One File to Another

8.6.2 ตัวอย่าง: นับจำนวนคำในไฟล์

การใช้การจัดการไฟล์เพื่อนับจำนวนคำในไฟล์ เป็นตัวอย่างการใช้งานจริงที่แสดงถึงความยืดหยุ่นของกระบวนการอ่านข้อมูลในภาษา Python โดยขั้นตอนนี้เริ่มจากการเปิดไฟล์เป้าหมายในโหมดอ่าน และอ่านเนื้อหาทั้งหมดโดยใช้เมธอด `read()` จากนั้นใช้เมธอด `split()` เพื่อแยกสตริงออกเป็นคำแต่ละคำ และใช้ฟังก์ชัน `len()` เพื่อหา

จำนวนคำทั้งหมด ตัวอย่างนี้แสดงให้เห็นว่า การจัดการไฟล์สามารถนำไปใช้กับการวิเคราะห์ข้อมูล ซึ่งเน้นถึงความสำคัญของการเข้าใจการดำเนินการกับไฟล์อย่างถ่องแท้เพื่อการเขียนโปรแกรมอย่างมีประสิทธิภาพ

ตัวอย่าง:

```
def count_words(file_name):
    with open(file_name, "r") as file:
        content = file.read()
        words = content.split()
        return len(words)

file_name = "example.txt"
print(f"The file {file_name} has {count_words(file_name)} words.")
```

Listing 8.22: Counting Words in a File

8.6.3 ตัวอย่าง: เขียนข้อมูลจากผู้ใช้ลงในไฟล์

การใช้การจัดการไฟล์เพื่อเขียนข้อมูลที่ผู้ใช้ป้อนลงในไฟล์ เป็นตัวอย่างที่แสดงการบูรณาการระหว่างการรับข้อมูลและการจัดการไฟล์ใน Python โดยกระบวนการนี้เริ่มจากการใช้ฟังก์ชัน `input()` เพื่อรับข้อความจากผู้ใช้ แล้วใช้เมธอด `write()` เพื่อเขียนข้อความลงในไฟล์ สามารถเลือกเปิดไฟล์ในโหมดเขียนหรือเพิ่มข้อมูลได้ตามต้องการ โดยการใช้คำสั่ง `with` ช่วยให้มั่นใจว่าไฟล์จะถูกปิดอย่างถูกต้องหลังจากเขียนข้อมูลเสร็จ ซึ่งช่วยป้องกันการสูญหายของข้อมูล ตัวอย่างนี้แสดงให้เห็นถึงวิธีการจัดเก็บข้อมูลที่สร้างขึ้นจากผู้ใช้อย่างมีประสิทธิภาพ ซึ่งเป็นทักษะสำคัญสำหรับการสร้างโปรแกรมแบบอินเทอร์แอคทีฟ

ตัวอย่าง:

```
# Writing user input to a file
with open("user_input.txt", "w") as file:
    while True:
        user_input = input("Enter text (type 'exit' to quit): ")
        if user_input.lower() == 'exit':
            break
        file.write(user_input + "\n")
```

Listing 8.23: Writing User Input to a File

บทที่ 8 โจทย์และแบบฝึกหัด: การจัดการไฟล์และการดำเนินการ I/O

8.1 การเขียนข้อมูลลงในไฟล์
เขียนโปรแกรม Python ที่เปิดไฟล์ใหม่ชื่อ `student_data.txt` *ในโหมดเขียน และเขียนข้อมูล: "Name: John Doe, Age: 20, Course: Computer Science"*

8.2 การอ่านและแสดงเนื้อหาในไฟล์
เขียนโปรแกรม Python ที่อ่านเนื้อหาจาก `student_data.txt` *และแสดงผลแต่ละบรรทัดบนหน้าจอ*

8.3 การเพิ่มข้อมูลลงในไฟล์ที่มีอยู่เดิม
เขียนโปรแกรม Python เพื่อเพิ่มข้อมูล "Grade: A" ลงในไฟล์ `student_data.txt` *โดยไม่ลบเนื้อหาเดิม*

8.4 การนับจำนวนบรรทัดในไฟล์
เขียนโปรแกรม Python เพื่อคำนวณจำนวนบรรทัดทั้งหมดในไฟล์ `student_data.txt` *และแสดงผล*

8.5 การคัดลอกข้อมูลจากไฟล์หนึ่งไปยังอีกไฟล์หนึ่ง
เขียนโปรแกรม Python เพื่อคัดลอกเนื้อหาจาก `student_data.txt` *ไปยังไฟล์ใหม่ชื่อ* `backup.txt`

8.6 การนับจำนวนคำในไฟล์
เขียนโปรแกรม Python ที่นับจำนวนคำทั้งหมดในไฟล์ `student_data.txt` *และแสดงผลรวม*

8.7 การเขียนข้อมูลจากผู้ใช้ลงในไฟล์
เขียนโปรแกรม Python ที่รับชื่อและอายุจากผู้ใช้ และบันทึกข้อมูลลงในไฟล์ `user_info.txt`

8.8 การอ่านและหาผลรวมของตัวเลขจากไฟล์
เขียนโปรแกรม Python ที่อ่านตัวเลขจากไฟล์ `numbers.txt` *(บรรทัดละตัวเลขหนึ่งค่า) และแสดงผลรวมของตัวเลขทั้งหมด*

8.9 การดึงที่อยู่อีเมลจากไฟล์
เขียนโปรแกรม Python ที่อ่านไฟล์ `contacts.txt` *ซึ่งเก็บอีเมลในแต่ละบรรทัด และจัดเก็บอีเมลทั้งหมดในลิสต์*

8.10 การบันทึกข้อผิดพลาดลงในไฟล์
เขียนโปรแกรม Python ที่รับข้อมูลจากผู้ใช้ และบันทึกข้อผิดพลาด (เช่น `ValueError`*) ลงในไฟล์* `error_log.txt`

ภาคผนวก

ภาคผนวกนี้ให้ทรัพยากรเสริม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งาน เพื่อเสริมสร้างความเข้าใจในแนวคิดที่ครอบคลุมในบทที่ 8

A8.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Python Programming: An Introduction to Computer Science* โดย John Zelle
- *Automate the Boring Stuff with Python* โดย Al Sweigart

บทเรียนออนไลน์:

- การจัดการไฟล์ใน Python - W3Schools
- การทำงานกับไฟล์ใน Python - Real Python

คอร์สเรียน:

- Coursera: โครงสร้างข้อมูลใน Python โดยมหาวิทยาลัยมิชิแกน
- Udemy: Python สำหรับ Data Science และ Machine Learning Bootcamp

A8.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). เอกสารประกอบ Python - การรับและส่งข้อมูล. ดึงข้อมูลจาก https://docs.python.org/3/tutorial/inputoutput.html
- Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook* (ฉบับที่ 3). สำนักพิมพ์ O'Reilly Media.

A8.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การดำเนินการไฟล์พื้นฐาน

- เขียนโปรแกรม Python เพื่อสร้างไฟล์ใหม่ และเขียนข้อความลงในไฟล์
- สร้างสคริปต์ Python เพื่ออ่านเนื้อหาในไฟล์ และแสดงผลบนหน้าจอ

แบบฝึกหัดที่ 2: การอ่านและเขียนไฟล์

- เขียนโปรแกรม Python เพื่อเพิ่มข้อความในไฟล์ที่มีอยู่ โดยไม่ลบเนื้อหาเดิม
- พัฒนาฟังก์ชันที่อ่านไฟล์ทีละบรรทัด และนับจำนวนบรรทัดทั้งหมด

แบบฝึกหัดที่ 3: การทำงานกับโหมดไฟล์ต่าง ๆ

- เขียนสคริปต์ Python ที่เปิดไฟล์ในโหมดไบนารีและอ่านเนื้อหา
- สร้างโปรแกรมที่เขียนลิสต์ของสตริงลงในไฟล์ โดยให้แต่ละสตริงอยู่คนละบรรทัด

แบบฝึกหัดที่ 4: การจัดการไฟล์ในสถานการณ์จริง

- เขียนฟังก์ชันที่รับชื่อไฟล์เป็นอาร์กิวเมนต์ และแสดงความถี่ของคำแต่ละคำในไฟล์
- สร้างสคริปต์ Python ที่อ่านไฟล์ CSV และแสดงผลลัพธ์ในรูปแบบตาราง

A8.4 ตัวอย่างการใช้งานจริง

ตัวอย่างที่ 1: การเขียนไฟล์

```python
# Writing to a file
with open("example.txt", "w") as file:
    file.write("Hello, World!\n")
    file.write("This is a sample file.\n")

# Reading from a file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

Listing 8.24: Writing to a file

ตัวอย่างที่ 2: การเพิ่มข้อมูลลงในไฟล์

```python
# Appending text to a file
with open("example.txt", "a") as file:
    file.write("Appending a new line.\n")

# Reading the updated file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

Listing 8.25: Appending text to a file

ตัวอย่างที่ 3: การอ่านไฟล์ทีละบรรทัด

```python
# Reading a file line by line
with open("example.txt", "r") as file:
    for line in file:
        print(line.strip())
```

Listing 8.26: Reading a file line by line

ตัวอย่างที่ 4: การนับจำนวนบรรทัดในไฟล์

```python
# Counting the number of lines in a file
def count_lines(file_name):
    with open(file_name, "r") as file:
        lines = file.readlines()
        return len(lines)

# Test the function
print("Number of lines:", count_lines("example.txt"))
```

Listing 8.27: Counting the number of lines in a file

ตัวอย่างที่ 5: การนับความถี่ของคำในไฟล์

```
# Function to count word frequency in a file
def word_frequency(file_name):
    with open(file_name, "r") as file:
        text = file.read()
        words = text.split()
        freq_dict = {}
        for word in words:
            if word in freq_dict:
                freq_dict[word] += 1
            else:
                freq_dict[word] = 1
    return freq_dict

# Test the function
print(word_frequency("example.txt"))
```

Listing 8.28: Function to count word frequency in a file

ตัวอย่างที่ 6: การอ่านไฟล์ CSV

```
import csv

# Reading a CSV file and printing its contents
def read_csv(file_name):
    with open(file_name, newline='') as csvfile:
        csvreader = csv.reader(csvfile)
        for row in csvreader:
            print(', '.join(row))

# Test the function with a sample CSV file
read_csv("sample.csv")
```

Listing 8.29: Reading a CSV file and printing its contents

บทที่ 9

การจัดการข้อผิดพลาด

เนื้อหานี้นำเสนอภาพรวมเกี่ยวกับการจัดการข้อผิดพลาด (Exception Handling) ในภาษา Python อย่างครบถ้วน ซึ่งครอบคลุมการใช้งานบล็อก `try-except-else-finally` พร้อมตัวอย่างการใช้งานจริงเพื่อให้เข้าใจการนำไปใช้ได้ชัดเจน การจัดการข้อผิดพลาดเป็นส่วนสำคัญในการเขียนโปรแกรม ช่วยให้นักพัฒนาสามารถคาดการณ์และรับมือกับข้อผิดพลาดที่อาจเกิดขึ้นระหว่างการทำงานของโปรแกรมได้ โดยการเข้าใจข้อผิดพลาดมาตรฐาน เช่น `SyntaxError`, `TypeError`, และ `ZeroDivisionError` จะช่วยให้สามารถเขียนโค้ดที่ทนทานต่อข้อผิดพลาดมากยิ่งขึ้น บล็อก `try-except` มีความจำเป็นในการทดสอบโค้ดและจัดการข้อผิดพลาด ในขณะที่ `else` จะทำงานก็ต่อเมื่อไม่มีข้อผิดพลาดเกิดขึ้น และ `finally` จะใช้สำหรับจัดการทรัพยากร เช่น การปิดไฟล์ ไม่ว่าจะเกิดข้อผิดพลาดหรือไม่ก็ตาม การรวมกันของบล็อกเหล่านี้ทำให้สามารถจัดการสถานการณ์ที่ซับซ้อนได้อย่างมีประสิทธิภาพ ตัวอย่างจริง เช่น การจัดการไฟล์หรือการตรวจสอบข้อมูลจากผู้ใช้ แสดงให้เห็นถึงการประยุกต์ใช้แนวคิดเหล่านี้ในสถานการณ์จริง บทนี้จะช่วยให้ผู้อ่านสามารถเขียนโค้ดที่มีความทนทานต่อข้อผิดพลาดมากยิ่งขึ้น เพิ่มความน่าเชื่อถือและความสามารถในการดูแลรักษาของโปรแกรม

9.1 บทนำสู่ข้อผิดพลาด

ข้อผิดพลาด (Exceptions) คือข้อผิดพลาดที่เกิดขึ้นระหว่างการทำงานของโปรแกรม ซึ่งขัดขวางการทำงานตามลำดับปกติของคำสั่ง ข้อผิดพลาดเหล่านี้อาจเกิดจากหลายสาเหตุ เช่น การป้อนข้อมูลไม่ถูกต้อง การหารด้วยศูนย์ หรือการเข้าถึงตำแหน่งของลิสต์ที่อยู่นอกขอบเขต เมื่อเกิดข้อผิดพลาดเช่นนี้ โปรแกรมอาจหยุดทำงานหรือแสดงผลที่ไม่คาดคิด อย่างไรก็ตาม Python มีระบบการจัดการข้อผิดพลาดที่มีประสิทธิภาพ ช่วยให้โปรแกรมสามารถทำงานต่อไปได้หรือจัดการข้อผิดพลาดนั้นอย่างสุภาพ ความสามารถนี้สำคัญมากสำหรับการสร้างซอฟต์แวร์ที่เชื่อถือได้ และสามารถรับมือกับสถานการณ์ที่ไม่คาดคิดได้อย่างเหมาะสม

แนวคิดสำคัญเกี่ยวกับการจัดการข้อผิดพลาด:

- **ข้อผิดพลาด (Exceptions)**: ความผิดพลาดระหว่างการทำงานของโปรแกรมที่ขัดจังหวะการทำงานปกติ
- **บล็อก Try**: โค้ดที่อาจก่อให้เกิดข้อผิดพลาดจะถูกวางไว้ใน `try`
- **บล็อก Except**: โค้ดที่ใช้จัดการข้อผิดพลาด หากเกิดขึ้น จะอยู่ใน `except`
- **บล็อก Else**: โค้ดที่จะทำงานหากไม่มีข้อผิดพลาดเกิดขึ้นใน `try`
- **บล็อก Finally**: โค้ดที่ทำงานเสมอ ไม่ว่าข้อผิดพลาดจะเกิดขึ้นหรือไม่ มักใช้สำหรับการทำความสะอาด เช่น ปิดไฟล์
- **การจับข้อผิดพลาดหลายประเภท**: ใช้หลาย `except` เพื่อจัดการข้อผิดพลาดต่างชนิด
- **การจับข้อผิดพลาดทั้งหมด**: ใช้คลาส `Exception` เพื่อจับข้อผิดพลาดทุกประเภท
- **การโยนข้อผิดพลาด**: ใช้คำสั่ง `raise` เพื่อสร้างข้อผิดพลาดตามเงื่อนไขที่กำหนด
- **ข้อผิดพลาดที่กำหนดเอง**: นิยามคลาสข้อผิดพลาดของตนเองเพื่อจัดการสถานการณ์เฉพาะ
- **การจัดการทรัพยากร**: ใช้ `finally` เพื่อให้แน่ใจว่าทรัพยากรถูกปิดหรือคืนค่า แม้เกิดข้อผิดพลาด
- **ข้อความแสดงข้อผิดพลาด**: การให้ข้อความที่มีประโยชน์เพื่อช่วยดีบักและเข้าใจสาเหตุของข้อผิดพลาด
- **แนวปฏิบัติที่ดี**: เขียนโค้ดที่สะอาด ดูแลรักษาง่าย และจัดการข้อผิดพลาดได้อย่างเหมาะสม

9.2 ประเภทของข้อผิดพลาดที่พบบ่อย

การเข้าใจประเภทของข้อผิดพลาดที่พบบ่อยถือเป็นสิ่งสำคัญในการจัดการข้อผิดพลาดอย่างมีประสิทธิภาพใน Python ข้อผิดพลาดเหล่านี้ช่วยระบุและแก้ไขปัญหาที่อาจเกิดขึ้นระหว่างการทำงานของโปรแกรม การทราบวิธีจัดการกับข้อผิดพลาดเหล่านี้จะช่วยให้โค้ดมีความน่าเชื่อถือและยืดหยุ่นมากขึ้น ต่อไปนี้คือตัวอย่างข้อผิดพลาดที่มักพบเป็นประจำซึ่งนักพัฒนา Python ควรรู้จัก:

- **SyntaxError**: เกิดเมื่อมีข้อผิดพลาดด้านไวยากรณ์ของโค้ด ทำให้ไม่สามารถแปลคำสั่งได้
- **TypeError**: เกิดเมื่อใช้ฟังก์ชันหรือการดำเนินการกับชนิดข้อมูลที่ไม่เหมาะสม เช่น การบวกตัวเลขกับสตริง

- **ValueError**: เกิดเมื่อฟังก์ชันได้รับค่าอาร์กิวเมนต์ที่ชนิดข้อมูลถูกต้องแต่ค่าผิด เช่น จำนวนลบที่คาดว่าควรเป็นบวก
- **IndexError**: เกิดเมื่อพยายามเข้าถึงตำแหน่งในลิสต์ที่อยู่นอกขอบเขต
- **KeyError**: เกิดเมื่อใช้คีย์ที่ไม่มีอยู่ในดิกชันนารี
- **ZeroDivisionError**: เกิดเมื่อพยายามหารตัวเลขด้วยศูนย์ ซึ่งทางคณิตศาสตร์ไม่สามารถทำได้

ตัวอย่างประกอบ

โค้ดในตัวอย่างนี้แสดงให้เห็นว่า Python สามารถจัดการกับข้อผิดพลาดเฉพาะเจาะจงได้อย่างไร เพื่อป้องกันไม่ให้โปรแกรมหยุดทำงานกะทันหันจากข้อผิดพลาด เช่น การหารด้วยศูนย์ โดย `except` จะช่วยจับข้อผิดพลาดและแสดงข้อความที่เข้าใจง่ายแก่ผู้ใช้ ซึ่งช่วยเพิ่มความแข็งแกร่งให้กับโปรแกรม

```
# Example: Handling division by zero
try:
    number = int(input("Please enter a number: "))
    result = 10 / number
    print("The result is:", result)
except ZeroDivisionError:
    print("Cannot divide by zero. Please enter a valid number.")
```

Listing 9.1: การจัดการข้อผิดพลาด ZeroDivisionError ด้วย try-except

กล่าวโดยสรุป โค้ดนี้แสดงกลไกหลักของการจัดการข้อผิดพลาดโดยใช้บล็อก `try` และ `except` โดยเน้นไปที่การจัดการข้อผิดพลาด `ZeroDivisionError` โดยเฉพาะ

9.3 บล็อก Try-Except

บล็อก `try` ช่วยให้คุณสามารถทดสอบโค้ดที่อาจเกิดข้อผิดพลาดได้ โดยครอบคลุมโค้ดที่อาจสร้างข้อผิดพลาดขึ้น หากเกิดข้อผิดพลาดภายในบล็อก `try` การควบคุมโปรแกรมจะถูกส่งไปยังบล็อก `except` ทันที ซึ่งจะมีโค้ดที่ใช้จัดการข้อผิดพลาดนั้น การใช้ `try` และ `except` ร่วมกันช่วยให้โปรแกรมไม่ล้มเมื่อเกิดข้อผิดพลาดที่ไม่คาดคิด และสามารถแสดงข้อความหรือจัดการสถานการณ์นั้นได้อย่างเหมาะสม โครงสร้างนี้มีความสำคัญอย่างยิ่งในการสร้างโปรแกรมที่มีความยืดหยุ่นและทนทาน

ไวยากรณ์:

```
try:
    # code that may raise an exception
except ExceptionType:
    # code to handle the exception
```

ตัวอย่าง:

```
# Example of common exceptions
try:
    x = 1 / 0  # ZeroDivisionError
except ZeroDivisionError as e:
    print(f"Error: {e}")

print("End of program")
#Output : Error: division by zero
```

Listing 9.2: Example of Common Exceptions

คำอธิบายตัวอย่าง: ข้อยกเว้นทั่วไป

โค้ด Python ด้านล่างนี้แสดงการจัดการข้อผิดพลาดโดยเฉพาะ `ZeroDivisionError` ซึ่งอธิบายเป็นขั้นตอนดังนี้:

- **บล็อก Try:**
  - ภายใน `try` มีโค้ดที่อาจสร้างข้อผิดพลาด เช่น `x = 1 / 0` ซึ่งเป็นการหารด้วยศูนย์
  - หากไม่มีการจัดการข้อผิดพลาด โปรแกรมจะล้มทันที
- **บล็อก Except:**
  - บล็อก `except ZeroDivisionError as e` จะทำหน้าที่จับข้อผิดพลาดนั้น
  - เมื่อข้อผิดพลาดเกิดขึ้น โปรแกรมจะไม่ล้ม แต่จะดำเนินการใน `except` แทน
  - `as e` จะเก็บวัตถุข้อผิดพลาดไว้ในตัวแปร `e`
- **การจัดการข้อผิดพลาด:**
  - ใน `except` จะพิมพ์ข้อความโดยใช้ `print(f"Error: {e}")` ซึ่งจะแสดง `division by zero`

ตัวอย่าง:

```
# Get the name of a file.
filename = input('Enter a filename: ')
try:
    # Open the file.
    infile = open(filename, 'r')
    # Read the file's contents.
    contents = infile.read()
    # Display the file's contents.
    print(contents)
    # Close the file.
    infile.close()
except IOError:
    print('An error occurred trying to read')
    print('the file', filename)

print("End of program")
```

Listing 9.3: Try-Execpt File Open Example

9.4 การจับข้อผิดพลาดหลายประเภท

คุณสามารถจับข้อผิดพลาดหลายประเภทได้โดยใช้บล็อก `except` หลายบล็อก ซึ่งแต่ละบล็อกจะจัดการข้อผิดพลาดเฉพาะเจาะจง วิธีนี้ช่วยให้คุณตอบสนองต่อข้อผิดพลาดต่าง ๆ ได้อย่างเหมาะสม ตัวอย่างเช่น บล็อกหนึ่งสามารถจับ `ValueError` ที่เกิดจากค่าที่ไม่ถูกต้อง ในขณะที่อีกบล็อกหนึ่งสามารถจัดการ `ZeroDivisionError` ที่เกิดจากการหารด้วยศูนย์ การมีบล็อก `except` หลายบล็อกช่วยให้โค้ดมีความแม่นยำ และยืดหยุ่นมากยิ่งขึ้น ทำให้โปรแกรมมีความน่าเชื่อถือสูงขึ้น

ตัวอย่าง:

```
try:
    value = int(input("Enter a number: "))
    result = 10 / value
except ValueError:
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero!")

print("End of program")
```

Listing 9.4: Catching Multiple Exceptions

9.5 การจับข้อผิดพลาดทั้งหมด

หากต้องการจับข้อผิดพลาดทุกรูปแบบ คุณสามารถใช้คลาส `Exception` โดยการวางบล็อก `except Exception` หลังบล็อกที่จับข้อผิดพลาดเฉพาะเจาะจง วิธีนี้ช่วยให้สามารถจัดการข้อผิดพลาดที่ไม่คาดคิดได้อย่างเหมาะสม อย่างไรก็ตาม ควรใช้ด้วยความระมัดระวัง เพราะอาจปิดบังข้อผิดพลาดสำคัญที่ควรถูกตรวจสอบหรือแก้ไขแยกต่างหาก

ตัวอย่าง:

```
try:
    value = int(input("Enter a number: "))
    result = 10 / value
except Exception as e:
    print(f"An error occurred: {e}")

print("End of program")
```

Listing 9.5: Catching All Exceptions

9.6 Else Block

```
try:
    value = int(input("Enter a number: "))
    result = 10 / value
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print(f"The result is {result}")

print("End of program")
```

Listing 9.6: Else Block

ตัวอย่างนี้แสดงให้เห็นการใช้ `else` เพื่อแยกโค้ดที่ทำงานเฉพาะเมื่อไม่มีข้อผิดพลาดใน `try` ทำให้โค้ดอ่านง่ายขึ้นและแยกการจัดการข้อผิดพลาดออกจากลอจิกปกติได้ชัดเจน

```
def divide(a, b):
    return a / b

a, b = map(int, input().split())
print(divide(a, b))
print("End of program")
```

Listing 9.7: Normal Divider Function

```python
def divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError as e:
        print("Exception:", e)
    else:
        return result

a, b = map(int, input().split())
print(divide(a, b))
print("End of program")
```

Listing 9.8: Divider Function with Exception Handler

9.7 Finally Block

```python
try:
    numerator = float(input("Enter the numerator: "))
    denominator = float(input("Enter the denominator: "))
    result = numerator / denominator
    print(f"The result is: {result}")
except ZeroDivisionError:
    print("Error: You cannot divide by zero.")
except ValueError:
    print("Error: Invalid input. Please enter numeric values.")
finally:
    print("Execution completed, whether an exception occurred or not.")

print("End of program")
```

Listing 9.9: Finally Block Example

โค้ดในตัวอย่างนี้ใช้ `finally` เพื่อให้มั่นใจว่าคำสั่งสุดท้ายจะทำงานเสมอไม่ว่าจะเกิดข้อผิดพลาดหรือไม่ ซึ่งเหมาะอย่างยิ่งสำหรับการทำความสะอาด (cleanup) เช่น ปิดไฟล์หรือปล่อยทรัพยากรระบบ

9.8 Combining Try-Except-Else-Finally

การรวมโครงสร้าง `try except else` และ `finally` เข้าไว้ด้วยกัน ทำให้สามารถจัดการข้อผิดพลาดได้อย่างครอบคลุม โดยเฉพาะการแยกความรับผิดชอบในแต่ละสถานการณ์ เช่น ตรวจจับข้อผิดพลาดใน `try` จัดการแต่ละข้อผิดพลาดใน `except` รันโค้ดปกติใน `else` และทำความสะอาดหรือปิดทรัพยากรใน `finally`

โครงสร้างไวยากรณ์:

```python
try:
    # code that may raise an exception
except ExceptionType:
    # code to handle the exception
else:
    # code to execute if no exceptions were raised
finally:
    # code to execute regardless of exceptions
```

ตัวอย่าง: การรวมการจัดการข้อผิดพลาดแบบครบถ้วน

```
try:
    value = int(input("Enter a number: "))
    result = 10 / value
except ValueError:
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print(f"The result is {result}")
finally:
    print("Execution completed.")
```

Listing 9.10: การรวมโครงสร้าง try-except-else-finally

9.9 Defining and Using a Custom Exception in Python

คุณ สามารถ สร้าง ข้อ ผิด พลาด แบบ กำหนด เอง (Custom Exception) โดย การ สร้าง คลาส ใหม่ ที่ สืบทอด จาก `Exception` ซึ่งช่วยให้สามารถตรวจจับเงื่อนไขเฉพาะทางได้ชัดเจนและสื่อความหมายมากขึ้น

ตัวอย่าง: การกำหนดและใช้งานข้อยกเว้นแบบกำหนดเอง

```
# Define a custom exception
class NegativeNumberError(Exception):
    def __init__(self, value):
        self.value = value
        super().__init__(f"Invalid input: {value} is a negative number")

# Function to check for positive numbers
def check_positive_number(num):
    if num < 0:
        raise NegativeNumberError(num)
    else:
        print(f"{num} is a valid positive number.")

# Using the function and handling the custom exception
try:
    number = int(input("Enter a positive number: "))
    check_positive_number(number)
except NegativeNumberError as e:
    print(e)
except ValueError:
    print("Error: Please enter a valid integer.")
finally:
    print("Program execution finished.")
```

Listing 9.11: Custom Exception Example

คำอธิบายตัวอย่าง:

- **1. การสร้างข้อยกเว้นใหม่:** คลาส `NegativeNumberError` ถูกสร้างขึ้นโดยสืบทอดจาก `Exception` เพื่อรองรับกรณีที่ผู้ใช้ป้อนค่าติดลบ

- **2. การตรวจสอบค่า:** ฟังก์ชัน `check_positive_number()` จะตรวจสอบว่าเป็นค่าบวก หากไม่ใช่จะ `raise` ข้อผิดพลาดนี้ขึ้นมา

- **3. การจัดการข้อผิดพลาด:** ใช้ `try-except` เพื่อดักจับทั้ง `NegativeNumberError` และ `ValueError`

- **4. บล็อก finally:** ใช้เพื่อแสดงข้อความว่าการทำงานของโปรแกรมสิ้นสุดแล้ว ไม่ว่าจะเกิดข้อผิดพลาดหรือไม่ก็ตาม

9.10 Practical Examples

ตัวอย่างการใช้งานจริงในบทนี้แสดงให้เห็นถึงการนำการจัดการข้อผิดพลาด (Exception Handling) มาใช้ในสถานการณ์จริง เช่น การเปิดและอ่านไฟล์ที่อาจไม่พบไฟล์ (FileNotFoundError) หรือการรับข้อมูลจากผู้ใช้ที่อาจไม่ถูกต้อง (ValueError) การใช้โครงสร้าง `try-except-else-finally` อย่างมีประสิทธิภาพช่วยให้โปรแกรมมีความเสถียร ไม่หยุดทำงานอย่างกะทันหัน และสามารถแจ้งเตือนผู้ใช้ด้วยข้อความที่เหมาะสม ทำให้โปรแกรมมีความน่าเชื่อถือและใช้งานได้ดีมากยิ่งขึ้น

9.10.1 Example: Handling File Operations

การจัดการข้อผิดพลาดระหว่างการเปิดไฟล์เป็นสิ่งสำคัญเพื่อป้องกันปัญหาที่อาจเกิดจากไฟล์ไม่มีอยู่จริง หรือเกิดข้อผิดพลาดระหว่างการอ่านไฟล์ การใช้ `try` เพื่อลองเปิดไฟล์, `except` เพื่อตรวจจับข้อผิดพลาด และ `finally` เพื่อให้มั่นใจว่าไฟล์จะถูกปิดอย่างเหมาะสม ช่วยให้การจัดการทรัพยากรและข้อความผิดพลาดแก่ผู้ใช้มีความปลอดภัยและเหมาะสม

```
def read_file(file_name):
    try:
        with open(file_name, "r") as file:
            content = file.read()
        return content
    except FileNotFoundError:
        print(f"The file {file_name} does not exist.")
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        print("File operation completed.")

file_content = read_file("example.txt")
```

Listing 9.12: Handling File Operations

9.10.2 Example: Input Validation

การรับค่าจากผู้ใช้เป็นอีกหนึ่งจุดที่อาจเกิดข้อผิดพลาดได้ เช่น การรับค่าตัวเลข แต่ผู้ใช้ป้อนข้อความหรือค่าที่ไม่เหมาะสม การใช้ `try-except` จะช่วยตรวจจับและแสดงข้อความเตือนที่เหมาะสม รวมถึงสามารถวนกลับเพื่อให้ผู้ใช้กรอกข้อมูลใหม่ได้จนกว่าจะถูกต้อง ซึ่งเป็นการเพิ่มความยืดหยุ่นและความน่าใช้ของโปรแกรม

```python
def get_positive_number():
    while True:
        try:
            number = int(input("Enter a positive number: "))
            if number <= 0:
                raise ValueError("The number is not positive.")
            return number
        except ValueError as e:
            print(f"Invalid input: {e}")

positive_number = get_positive_number()
print(f"You entered: {positive_number}")
```

Listing 9.13: Input Validation

บทที่ 9 โจทย์และแบบฝึกหัด: การจัดการข้อผิดพลาด (Exception Handling)

9.1 เข้าใจการจัดการข้อผิดพลาดเบื้องต้น
เขียนโปรแกรมที่ให้ผู้ใช้ป้อนชื่อไฟล์และพยายามเปิดอ่านเนื้อหา หากไม่พบไฟล์ ให้จัดการข้อผิดพลาด `FileNotFoundError` *และแสดงข้อความที่เข้าใจง่าย*

9.2 จัดการข้อผิดพลาดหลายประเภท
เขียนโปรแกรมที่ให้ผู้ใช้ป้อนจำนวนเต็มสองค่า แล้วทำการหารกัน จัดการข้อผิดพลาด `ValueError` *หากป้อนค่าที่ไม่ใช่ตัวเลข และ* `ZeroDivisionError` *หากหารด้วยศูนย์*

9.3 ใช้ else กับ try-except
เขียนโปรแกรมที่พยายามแปลงข้อความที่ผู้ใช้ป้อนเป็นจำนวนเต็ม หากสำเร็จให้พิมพ์ค่าที่แปลง และใช้บล็อก `else` *เพื่อแสดงข้อความว่าแปลงสำเร็จ*

9.4 การจัดการข้อผิดพลาดในฟังก์ชัน
กำหนดฟังก์ชันชื่อ `divide_numbers` *ที่รับพารามิเตอร์สองตัว และคืนค่าผลหาร โดยใช้ try-except จัดการข้อผิดพลาดภายในฟังก์ชัน*

9.5 สร้างและจัดการ Custom Exception
สร้างคลาสข้อผิดพลาดชื่อ `NegativeNumberError` *แล้วเขียนฟังก์ชัน* `check_positive` *ที่จะ raise ข้อผิดพลาดนี้เมื่อรับค่าติดลบ และจัดการข้อผิดพลาดนั้นในโปรแกรมหลัก*

9.6 จัดการข้อผิดพลาดหลายแบบในการเปิดไฟล์
เขียนโปรแกรมที่เปิดไฟล์ อ่านเนื้อหา และพิมพ์ออกทางหน้าจอ พร้อมจัดการข้อผิดพลาด `FileNotFoundError` *และ* `PermissionError` *ด้วยข้อความเฉพาะ*

9.7 การจัดการทรัพยากรด้วย finally
สร้างโปรแกรมที่พยายามเปิดและอ่านไฟล์ โดยใช้บล็อก `finally` *เพื่อให้แน่ใจว่าไฟล์จะถูกปิดทุกกรณีไม่ว่าจะเกิดข้อผิดพลาดหรือไม่ก็ตาม*

9.8 ตรวจสอบอินพุตด้วย Custom Exception
เขียนโปรแกรมที่ให้ผู้ใช้ป้อนจำนวนเต็มบวก หากไม่เป็นไปตามเงื่อนไขให้ raise ข้อผิดพลาด `NotPositiveError` *และจัดการข้อผิดพลาดนี้ให้เหมาะสม*

9.9 ใช้ try-except-else-finally ร่วมกัน
สร้างโปรแกรมที่ให้ผู้ใช้ป้อนตัวเลขสองตัว แล้วทำการหาร โดยใช้โครงสร้าง `try-except-else-finally` *ครบทุกส่วน*

9.10 จัดการข้อผิดพลาดในลูป
เขียนโปรแกรมที่ให้ผู้ใช้ป้อนตัวเลขจนกว่าจะเป็นจำนวนเต็มบวก โดยใช้ exception handling ภายในลูปเพื่อรับมือกับข้อมูลที่ไม่ถูกต้อง

ภาคผนวก

ภาคผนวกนี้นำเสนอแหล่งข้อมูลเพิ่มเติม อ้างอิง แบบฝึกหัด และตัวอย่างจริง เพื่อเสริมความเข้าใจของคุณเกี่ยวกับแนวคิดในบทที่ 9

A9.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Python Programming: An Introduction to Computer Science* โดย John Zelle
- *Fluent Python* โดย Luciano Ramalho

บทเรียนออนไลน์:

- การจัดการข้อผิดพลาดใน Python - W3Schools
- การจัดการข้อผิดพลาดใน Python - Real Python

คอร์สออนไลน์:

- Coursera: Python for Everybody โดยมหาวิทยาลัยมิชิแกน
- Udemy: The Python Mega Course: สร้างแอปพลิเคชันจริง 10 ตัวอย่าง

A9.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). เอกสาร Python - ข้อผิดพลาดและข้อยกเว้น จาก https://docs.python.org/3/tutorial/errors.html
- Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook* (ฉบับที่ 3). สำนักพิมพ์ O'Reilly Media.

A9.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การจัดการข้อผิดพลาดพื้นฐาน

- เขียนโปรแกรม Python เพื่อให้ผู้ใช้ป้อนตัวเลข และพิมพ์ค่ากลับด้าน (reciprocal) ของตัวเลขนั้น โดยจัดการกรณีที่ป้อนเลขศูนย์
- เขียนสคริปต์ที่อ่านจำนวนเต็มจากผู้ใช้ และแสดง `ValueError` หากป้อนค่าไม่ถูกต้อง

แบบฝึกหัดที่ 2: การจัดการข้อผิดพลาดหลายประเภท

- เขียนฟังก์ชัน Python ที่เปิดไฟล์และอ่านเนื้อหา พร้อมจัดการกรณีไม่พบไฟล์และไม่มีสิทธิ์เข้าถึง
- เขียนสคริปต์ที่รับค่าตัวเลข 2 ตัวแล้วหารกัน พร้อมจัดการข้อผิดพลาดจากการหารด้วยศูนย์และการป้อนค่าผิดประเภท

แบบฝึกหัดที่ 3: การใช้ else และ finally

- เขียนโปรแกรม Python ที่เขียนข้อมูลลงไฟล์ พร้อมใช้ `finally` เพื่อปิดไฟล์ทุกกรณี

- เขียนสคริปต์ที่พยายามเชื่อมต่อกับฐานข้อมูล และใช้ `else` แสดงข้อความสำเร็จหากไม่พบข้อผิดพลาด

แบบฝึกหัดที่ 4: ข้อยกเว้นแบบกำหนดเอง (Custom Exceptions)

- สร้างข้อยกเว้นแบบกำหนดเองชื่อ `NegativeNumberError` ซึ่งจะเกิดขึ้นเมื่อฟังก์ชันได้รับค่าติดลบ
- เขียนสคริปต์ที่ใช้ข้อยกเว้นดังกล่าว เพื่อจัดการการป้อนค่าที่ไม่ถูกต้องในฟังก์ชันที่คำนวณค่ารากที่สอง

A9.4 ตัวอย่างการใช้งานจริง

ตัวอย่างที่ 1: การจัดการข้อผิดพลาดเบื้องต้น

```
try:
    number = int(input("Enter a number: "))
    reciprocal = 1 / number
    print("The reciprocal is:", reciprocal)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
```

Listing 9.14: Handling division by zero exception

ตัวอย่างที่ 2: การจัดการข้อผิดพลาดหลายประเภท

```
def read_file(file_name):
    try:
        with open(file_name, 'r') as file:
            content = file.read()
            print(content)
    except FileNotFoundError:
        print("Error: File not found.")
    except PermissionError:
        print("Error: Permission denied.")

read_file("example.txt")
```

Listing 9.15: Function to read a file and handle exceptions

ตัวอย่างที่ 3: การใช้ else และ finally

```
try:
    file = open("example.txt", "w")
    file.write("Hello, World!")
except IOError:
    print("Error: Unable to write to file.")
else:
    print("Write operation successful.")
finally:
    file.close()
    print("File closed.")
```

Listing 9.16: Writing to a file with finally block to ensure file is closed

ตัวอย่างที่ 4: การใช้ข้อยกเว้นแบบกำหนดเอง

ในส่วนนี้เราจะสำรวจตัวอย่างขั้นสูงของการจัดการข้อผิดพลาดใน Python ซึ่งรวมถึง:

- การจัดการข้อผิดพลาดหลายประเภท เช่น `FileNotFoundError`, `ValueError` ฯลฯ
- การนิยามและเรียกใช้ข้อยกเว้นแบบกำหนดเองเมื่อตรวจพบเลขติดลบจากไฟล์
- การใช้ `finally` เพื่อให้มั่นใจว่าทรัพยากรถูกปิดแม้เกิดข้อผิดพลาด

```python
class NegativeNumberError(Exception):
    def __init__(self, number):
        super().__init__(f"Negative number found: {number}")
        self.number = number

def process_numbers(file_name):
    try:
        infile = open(file_name, 'r')
        total = 0
        for line in infile:
            number = float(line.strip())
            if number < 0:
                raise NegativeNumberError(number)
            total += number
        print(f"Total of all numbers: {total}")
    except FileNotFoundError:
        print(f"Error: The file '{file_name}' does not exist.")
    except ValueError:
        print("Error: The file contains invalid (non-numeric) data.")
    except NegativeNumberError as e:
        print(e)
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
    finally:
        try:
            infile.close()
            print("File closed.")
        except NameError:
            print("File was never opened, nothing to close.")

file_name = "numbers.txt"
process_numbers(file_name)
```

Listing 9.17: Advanced Exception Handling Example in Python

ผลลัพธ์ตัวอย่าง

กรณีที่ 1: ไม่พบไฟล์

```
Error: The file 'numbers.txt' does not exist.
File was never opened, nothing to close.
```

กรณีที่ 2: ไฟล์มีข้อมูลไม่ใช่ตัวเลข

```
Error: The file contains invalid (non-numeric) data.
File closed.
```

กรณีที่ 3: พบเลขติดลบ

```
Negative number found: -25.0
File closed.
```

กรณีที่ 4: ข้อมูลถูกต้องทั้งหมด

```
Total of all numbers: 100.5
File closed.
```

แหล่งข้อมูล อ้างอิง แบบฝึกหัด และตัวอย่างเหล่านี้มีวัตถุประสงค์เพื่อเพิ่มความเข้าใจในแนวคิดที่นำเสนอในบทที่ 9 และส่งเสริมให้นักเรียนฝึกฝนอย่างต่อเนื่อง

บทที่ 10

ไลบรารีและโมดูลของภาษาไพธอน

บทนี้นำเสนอภาพรวมที่ครบถ้วนและละเอียดเกี่ยวกับไลบรารีและโมดูลของภาษาไพธอน ซึ่งเป็นทรัพยากรสำคัญทั้งสำหรับผู้เริ่มต้นและผู้ที่มีประสบการณ์ในการเขียนโปรแกรม โดยเริ่มจากความเข้าใจพื้นฐานของโมดูลและไลบรารี อธิบายถึงวิธีการจัดระเบียบโค้ดให้เป็นส่วนย่อยที่จัดการได้ง่าย และช่วยให้สามารถนำโค้ดกลับมาใช้ใหม่ในโปรแกรมต่าง ๆ ได้ บทนี้จะกล่าวถึงแง่มุมในทางปฏิบัติของการนำเข้าโมดูล โดยครอบคลุมเทคนิคต่าง ๆ เช่น การนำเข้าทั้งหมด การนำเข้าเฉพาะบางส่วน การใช้ชื่อย่อ และการนำเข้าทุกส่วนจากโมดูล

บทนี้เน้นไลบรารีมาตรฐานของ Python โดยแสดงโมดูลที่ใช้กันบ่อย เช่น `os`, `sys`, `datetime`, `random`, และ `json` โดยแต่ละโมดูลจะอธิบายพร้อมตัวอย่างการใช้งานจริง เพื่อแสดงให้เห็นถึงหน้าที่และวิธีการใช้สำหรับการจัดการงานต่าง ๆ ที่หลากหลาย

นอกจากนี้ ยังแนะนำไลบรารีภายนอกยอดนิยม เช่น `NumPy`, `Pandas`, `Matplotlib`, `Requests`, และ `Flask` โดยกล่าวถึงความสำคัญ วิธีการติดตั้ง และให้ตัวอย่างการใช้งานเพื่อให้ผู้อ่านได้ฝึกฝนจากประสบการณ์จริง

ในส่วนของตัวอย่างเชิงปฏิบัติ บทนี้จะรวมการใช้หลายโมดูลร่วมกันในการทำงานที่ซับซ้อน เพื่อเสริมสร้างความเข้าใจของผู้อ่านว่าโมดูลต่าง ๆ สามารถทำงานร่วมกันได้อย่างไรเพื่อบรรลุเป้าหมายที่เฉพาะเจาะจง บทนี้จะช่วยให้ผู้อ่านมีความรู้และทักษะในการใช้งานไลบรารีและโมดูลของ Python ได้อย่างมีประสิทธิภาพในโครงการเขียนโปรแกรมของตน

10.1 บทนำสู่โมดูลและไลบรารี

ในภาษาไพธอน โมดูลและไลบรารีมีบทบาทสำคัญในการจัดระเบียบและนำโค้ดกลับมาใช้ใหม่ พวกมันช่วยแบ่งโปรแกรมที่ซับซ้อนให้กลายเป็นส่วนย่อยที่จัดการได้ง่ายขึ้น ทำให้การพัฒนามีประสิทธิภาพและเป็นระบบมากยิ่งขึ้น ไลบรารีมาตรฐานของ Python มีคลังโค้ดสำเร็จรูปมากมายสำหรับการทำงานหลากหลายรูปแบบ และยังสามารถขยายขีดความสามารถเพิ่มเติมได้ด้วยไลบรารีของบุคคลที่สาม การเข้าใจวิธีใช้โมดูลและไลบรารีจึงเป็นสิ่งจำเป็นในการเขียนโปรแกรมที่สะอาด ดูแลรักษาง่าย และมีประสิทธิภาพ

การเรียนรู้เกี่ยวกับโมดูลและไลบรารีสามารถช่วยเพิ่มทักษะและประสิทธิภาพในการเขียนโปรแกรมได้อย่างมาก มันช่วยให้สามารถจัดระเบียบโค้ดได้ดีขึ้น และมอบเครื่องมือที่ทรงพลังในการแก้ไขงานที่ซับซ้อน ทำให้การเขียนโปรแกรมเป็นเรื่องที่สนุกและมีประสิทธิภาพยิ่งขึ้น

10.1.1 โมดูลคืออะไร?

โมดูลคือไฟล์ที่มีโค้ดภาษาไพธอนอยู่ภายใน โดยโค้ดเหล่านี้สามารถประกอบด้วยฟังก์ชัน คลาส และตัวแปร ซึ่งช่วยให้คุณสามารถจัดระเบียบงานเขียนโปรแกรมให้เป็นส่วน ๆ ที่จัดการได้ง่ายขึ้น ลองนึกว่าโมดูลเป็นเหมือนกล่องเครื่องมือ เช่นเดียวกับที่คุณอาจมีกล่องเครื่องมือสำหรับซ่อมแซมสิ่งของภายในบ้าน โมดูลก็เป็นชุดของเครื่องมือ (ในรูปแบบของ

โค้ด) ที่สามารถนำมาใช้ในการทำงานเฉพาะอย่างในโปรแกรมของคุณ การใช้โมดูลจะช่วยให้สามารถแบ่งโปรแกรมออกเป็นชิ้นส่วนย่อย ทำให้โค้ดอ่านง่าย เข้าใจง่าย และดูแลรักษาได้ง่ายขึ้น นอกจากนี้ โมดูลยังช่วยให้สามารถนำโค้ดกลับมาใช้ใหม่ได้ โดยสามารถใช้ฟังก์ชันหรือคลาสเดียวกันกับหลายโปรแกรมโดยไม่ต้องเขียนใหม่ทุกครั้ง

แนวคิดสำคัญของโมดูล:

- **คำนิยาม**: ไฟล์ที่มีโค้ด Python เช่น ฟังก์ชัน คลาส ตัวแปร
- **การจัดระเบียบ**: ช่วยจัดโค้ดให้อยู่ในส่วนที่จัดการได้ง่าย
- **การนำกลับมาใช้**: ใช้งานโค้ดซ้ำได้ในหลายโปรแกรม
- **การนำเข้า**: ใช้คำสั่ง `import` เพื่อนำเข้าโมดูล
- **เนมสเปซ**: โมดูลสร้างเนมสเปซแยกต่างหากเพื่อหลีกเลี่ยงการชนกันของชื่อ

10.1.2 ไลบรารีคืออะไร?

ไลบรารีคือชุดของโมดูลที่เกี่ยวข้องกันซึ่งให้บริการเฉพาะด้าน ไลบรารีมาตรฐานของ Python เป็นตัวอย่างที่ดี ซึ่งมีชุดของโมดูลที่มาพร้อมกับภาษา ครอบคลุมฟังก์ชันหลากหลาย ตั้งแต่การคำนวณพื้นฐานและการจัดการสตริง ไปจนถึงงานที่ซับซ้อนกว่า เช่น การพัฒนาเว็บและการวิเคราะห์ข้อมูล ไลบรารีช่วยประหยัดเวลาและแรงงานโดยให้โค้ดที่เขียนและทดสอบไว้ล่วงหน้า ซึ่งสามารถนำมาใช้ในการพัฒนาโปรแกรมโดยไม่ต้องเริ่มต้นใหม่ทุกครั้ง

นอกจากไลบรารีมาตรฐานแล้ว ยังมีไลบรารีจากบุคคลที่สามอีกจำนวนมาก ซึ่งพัฒนาโดยนักพัฒนาอื่น ๆ และสามารถติดตั้งเพิ่มเติมได้ง่ายเพื่อขยายขีดความสามารถของ Python ยกตัวอย่างเช่น `NumPy` และ `Pandas` ที่ได้รับความนิยมในงานคำนวณเชิงวิทยาศาสตร์และการวิเคราะห์ข้อมูล หรือ `Flask` และ `Django` ที่ใช้ในการพัฒนาเว็บ

แนวคิดสำคัญของไลบรารี:

- **คำนิยาม**: ชุดของโมดูลที่เกี่ยวข้องกัน
- **ไลบรารีมาตรฐาน**: มาพร้อมกับ Python ให้ฟังก์ชันหลากหลาย
- **ไลบรารีจากบุคคลที่สาม**: พัฒนาเพิ่มเติมโดยชุมชน สามารถติดตั้งผ่าน `pip`
- **การนำโค้ดกลับมาใช้**: มีโค้ดที่เขียนและทดสอบไว้แล้วสำหรับงานทั่วไป
- **การขยายขีดความสามารถ**: ช่วยขยายขอบเขตการใช้งานของ Python
- **ตัวอย่าง**: `NumPy` สำหรับการคำนวณเชิงตัวเลข, `Pandas` สำหรับการวิเคราะห์ข้อมูล, `Flask` สำหรับการพัฒนาเว็บ

10.2 การนำเข้าโมดูล

การนำเข้าโมดูลในภาษาไพธอนช่วยให้คุณสามารถเข้าถึงฟังก์ชัน คลาส และตัวแปรที่กำหนดไว้ในไฟล์อื่น ๆ ซึ่งช่วยเพิ่มความสามารถในการนำโค้ดกลับมาใช้ใหม่และจัดระเบียบโค้ดได้ดีขึ้น คุณสามารถใช้คำสั่ง `import` เพื่อนำทรัพยากรภายนอกเหล่านี้เข้ามาในโปรแกรมของคุณ ทำให้สามารถใช้งานฟังก์ชันที่ซับซ้อนได้โดยไม่ต้องเขียนโค้ดทั้งหมดขึ้นมาใหม่

10.2.1 การนำเข้าโมดูลทั้งหมด

คุณสามารถนำเข้าโมดูลทั้งหมดโดยใช้คำสั่ง `import` ซึ่งจะทำให้คุณสามารถเข้าถึงฟังก์ชัน คลาส และตัวแปรทั้งหมดที่กำหนดไว้ในโมดูลนั้น วิธีนี้เหมาะเมื่อคุณต้องการใช้หลายองค์ประกอบจากโมดูลเดียวกัน หรือเมื่อคุณต้องการให้โค้ดมีความกระชับและเป็นระเบียบมากขึ้น การนำเข้าโมดูลทั้งหมดจะช่วยให้คุณสามารถอ้างอิงองค์ประกอบต่าง ๆ โดยใช้ชื่อของโมดูล ซึ่งช่วยรักษาความชัดเจนและหลีกเลี่ยงความขัดแย้งของชื่อในโค้ด

ตัวอย่าง:

```
import math

print(math.sqrt(16))  # Output: 4.0
print(math.pi)        # Output: 3.141592653589793
```

Listing 10.1: Importing an Entire Module

10.2.2 การนำเข้าส่วนเฉพาะจากโมดูล

คุณสามารถนำเข้าเฉพาะฟังก์ชัน คลาส หรือค่าคงที่บางรายการจากโมดูลโดยใช้ไวยากรณ์ `from ... import` วิธีนี้เหมาะอย่างยิ่งเมื่อคุณต้องการใช้งานเพียงบางส่วนของโมดูล และไม่ต้องการโหลดสิ่งที่ไม่จำเป็น ช่วยให้โค้ดมีประสิทธิภาพมากขึ้น เมื่อใช้วิธีนี้ คุณสามารถเรียกใช้องค์ประกอบที่นำเข้าได้โดยไม่ต้องใส่ชื่อโมดูลนำหน้า ซึ่งทำให้โค้ดดูสะอาดและอ่านง่ายยิ่งขึ้น ขณะเดียวกันก็ยังสามารถควบคุมสิ่งที่นำเข้าได้อย่างแม่นยำ

```
from math import sqrt, pi

print(sqrt(16))  # Output: 4.0
print(pi)        # Output: 3.141592653589793
```

Listing 10.2: Importing Specific Items from a Module

10.2.3 การนำเข้าโมดูลด้วยชื่อย่อ

คุณสามารถตั้งชื่อย่อให้กับโมดูลได้โดยใช้คำสำคัญ `as` ซึ่งช่วยให้สามารถอ้างอิงโมดูลได้สะดวกขึ้น โดยเฉพาะอย่างยิ่งในกรณีที่ชื่อของโมดูลยาวหรือใช้งานไม่สะดวก การตั้งชื่อย่อช่วยทำให้โค้ดกระชับ อ่านง่าย และดูแลรักษาได้ง่ายขึ้น โดยเฉพาะอย่างยิ่งในโครงการขนาดใหญ่หรือการทำงานร่วมกับผู้อื่น

```
import numpy as np

array = np.array([1, 2, 3, 4])
print(array)  # Output: [1 2 3 4]
```

Listing 10.3: Importing a Module with an Alias

10.2.4 การนำเข้าทุกองค์ประกอบจากโมดูล

คุณสามารถนำเข้าทุกองค์ประกอบจากโมดูลได้โดยใช้ไวยากรณ์ `from ... import *` อย่างไรก็ตาม วิธีนี้มักไม่แนะนำให้ใช้ เพราะอาจทำให้เกิดความขัดแย้งของชื่อ (namespace conflict) เมื่อองค์ประกอบที่นำเข้ามาทับซ้อน

กับตัวแปรหรือฟังก์ชันที่มีอยู่ในโค้ด วิธีนี้ยังทำให้ติดตามแหล่งที่มาของฟังก์ชันหรือค่าต่าง ๆ ได้ยากขึ้น ซึ่งอาจลดความสามารถในการอ่านและเพิ่มความเสี่ยงต่อข้อผิดพลาด โดยเฉพาะในโปรเจกต์ขนาดใหญ่

```
from math import *

print(sqrt(16))  # Output: 4.0
print(pi)        # Output: 3.141592653589793
```

Listing 10.4: Importing All Items from a Module

10.3 ภาพรวมของไลบรารีมาตรฐานของ Python

ไลบรารีมาตรฐานของ Python เป็นชุดรวมของโมดูลและแพ็กเกจที่มาพร้อมกับตัวภาษา ซึ่งครอบคลุมเครื่องมือและฟังก์ชันการทำงานมากมายสำหรับการจัดการงานต่าง ๆ ตั้งแต่การดำเนินการพื้นฐานในการเขียนโปรแกรม ไปจนถึงการจัดการข้อมูลที่ซับซ้อนและการพัฒนาเว็บ ไลบรารีนี้มีโมดูลสำหรับการทำงานกับไฟล์ ระบบเครือข่าย การจัดการข้อมูลแบบอนุกรม และอื่น ๆ อีกมากมาย

แหล่งทรัพยากรอันหลากหลายนี้ช่วยให้นักพัฒนาสามารถทำงานเขียนโปรแกรมทั่วไปได้โดยไม่ต้องติดตั้งซอฟต์แวร์เพิ่มเติม การใช้ไลบรารีมาตรฐานช่วยประหยัดเวลาและแรงงาน เนื่องจากโมดูลที่ใช้งานได้รับการทดสอบมาเป็นอย่างดีและทำงานได้อย่างราบรื่นกับภาษานี้ ไลบรารีภายในเหล่านี้เหมาะทั้งสำหรับผู้เริ่มต้นและผู้เชี่ยวชาญ เพราะช่วยให้การพัฒนาโปรแกรมเป็นไปอย่างมีประสิทธิภาพและสม่ำเสมอ โดยลดการพึ่งพาไลบรารีจากบุคคลที่สามสำหรับงานพื้นฐาน

10.4 โมดูลที่ใช้งานบ่อยในไลบรารีมาตรฐาน

ไลบรารีมาตรฐานของ Python ประกอบด้วยโมดูลที่ใช้งานบ่อยหลายรายการ ซึ่งให้ฟังก์ชันที่จำเป็นสำหรับงานเขียนโปรแกรมประจำวัน นักพัฒนานิยมใช้โมดูลเหล่านี้เนื่องจากความเสถียรและความสะดวกในการใช้งาน ตัวอย่างเช่น โมดูล `os` สำหรับการทำงานร่วมกับระบบปฏิบัติการ โมดูล `sys` สำหรับเข้าถึงพารามิเตอร์ของระบบ โมดูล `datetime` สำหรับการจัดการวันที่และเวลา และโมดูล `json` สำหรับการจัดการข้อมูล JSON

10.4.1 โมดูล os

โมดูล `os` ให้ฟังก์ชันหลากหลายสำหรับการทำงานร่วมกับระบบปฏิบัติการ เช่น การอ่านหรือเขียนไฟล์ การจัดการเส้นทางของไฟล์และไดเรกทอรี และการเข้าถึงตัวแปรแวดล้อม มันมีอินเทอร์เฟซที่ทรงพลังสำหรับการเรียกคำสั่งของ shell การจัดการระดับระบบ และการดึงข้อมูลเฉพาะของระบบ คุณสามารถสร้างโปรแกรม Python ที่มีความยืดหยุ่นและรองรับหลายแพลตฟอร์มได้ด้วยโมดูลนี้

```
import os

print(os.name)      # Output: posix (on Unix-like systems), nt (on Windows)
print(os.getcwd()) # Output: Current working directory
os.mkdir("new_directory")  # Create a new directory
```

Listing 10.5: os Module

10.4.2 โมดูล sys

โมดูล `sys` ให้การเข้าถึงตัวแปรและฟังก์ชันต่าง ๆ ที่ใช้ในการโต้ตอบกับตัวแปลภาษา Python ช่วยให้คุณสามารถควบค่าสภาพแวดล้อมระหว่างการทำงานได้ เช่น การจัดการอาร์กิวเมนต์ของบรรทัดคำสั่ง ควบคุมพฤติกรรมของตัวแปลภาษา และเข้าถึงอินพุต/เอาต์พุตมาตรฐาน รวมถึงการจัดการเส้นทางการค้นหาโมดูล โมดูลนี้เป็นเครื่องมือสำคัญในการปรับแต่งและควบคุมพฤติกรรมของโปรแกรม

```
import sys

print(sys.version)  # Output: Python version
sys.exit()          # Exit the program
```

Listing 10.6: Example of sys Module

10.4.3 โมดูล datetime

โมดูล `datetime` ให้คลาสสำหรับการจัดการวันที่และเวลา ช่วยให้คุณสามารถจัดการกับวันที่ เวลา และช่วงเวลาได้อย่างยืดหยุ่นและมีประสิทธิภาพ โมดูลนี้มีคลาส เช่น `date`, `time`, `datetime`, และ `timedelta` สำหรับการแทนค่าและการคำนวณเกี่ยวกับวันและเวลา คุณสามารถจัดรูปแบบวันที่ ทำเลขคณิตกับเวลา และจัดการโซนเวลาได้อย่างสะดวก จึงเป็นประโยชน์อย่างมากสำหรับแอปพลิเคชันที่ต้องการการจัดการเวลาที่แม่นยำ

```
import datetime

now = datetime.datetime.now()
print(now)  # Output: Current date and time

date = datetime.date(2024, 1, 1)
print(date)  # Output: 2024-01-01
```

Listing 10.7: Example of datetime Module

10.4.4 โมดูล random

โมดูล `random` ให้ฟังก์ชันสำหรับสร้างตัวเลขสุ่มและการดำเนินการแบบสุ่ม ซึ่งเหมาะสำหรับงานที่ต้องการความไม่แน่นอนหรือพฤติกรรมเชิงความน่าจะเป็น เช่น การสุ่มจำนวนเต็ม การสุ่มตัวเลขทศนิยม และการเลือกองค์ประกอบจากลิสต์แบบสุ่ม นอกจากนี้ยังสามารถสับลำดับข้อมูล สุ่มตัวอย่าง และสร้างค่าตามการแจกแจงความน่าจะเป็นต่าง ๆ โมดูลนี้ถูกใช้อย่างแพร่หลายในงานจำลอง เกม ความปลอดภัย และการสุ่มตัวอย่างทางสถิติ

```
import random

print(random.randint(1, 10))  # Output: Random integer between 1 and 10
print(random.choice(['apple', 'banana', 'cherry']))  # Output: Random
    choice from the list
```

Listing 10.8: Example of random Module

10.4.5 โมดูล json

โมดูล `json` ให้ฟังก์ชันสำหรับทำงานกับข้อมูล JSON โดยช่วยให้สามารถเข้ารหัสและถอดรหัสออบเจกต์ JSON ได้อย่างง่ายดาย สามารถแปลงโครงสร้างข้อมูลของ Python เช่น ดิกชันนารีและลิสต์ให้กลายเป็นสตริง JSON และในทางกลับกันได้อย่างรวดเร็ว ความสามารถนี้สำคัญมากสำหรับแอปพลิเคชันที่มีการแลกเปลี่ยนข้อมูลระหว่างโปรแกรม Python กับบริการเว็บหรือระบบที่ใช้ JSON อื่น ๆ โมดูล `json` ช่วยให้การจัดการข้อมูล JSON มีประสิทธิภาพและเชื่อถือได้ ซึ่งเป็นเครื่องมือสำคัญสำหรับการพัฒนาเว็บยุคใหม่และการประมวลผลข้อมูล

```
import json

data = {"name": "Alice", "age": 25}
json_str = json.dumps(data)
print(json_str)  # Output: JSON string

parsed_data = json.loads(json_str)
print(parsed_data)  # Output: Python dictionary
print(parsed_data["name"])  # Output: Alice
print(parsed_data["age"])  # Output: 25
```

Listing 10.9: Example of json Module

10.5 ภาพรวมของไลบรารีจากบุคคลที่สาม

ไลบรารีจากบุคคลที่สามช่วยขยายความสามารถของภาษา Python ให้เหนือกว่าไลบรารีมาตรฐาน โดยมอบเครื่องมือและฟังก์ชันเฉพาะทางสำหรับการใช้งานที่หลากหลาย ไลบรารีเหล่านี้สามารถหาได้จาก Python Package Index (PyPI) ซึ่งเป็นแหล่งรวมซอฟต์แวร์ Python ที่ครอบคลุม นักพัฒนาสามารถค้นหา ดาวน์โหลด และติดตั้งไลบรารีเหล่านี้ได้อย่างง่ายดายเพื่อเพิ่มศักยภาพให้กับโปรเจกต์ของตน

เครื่องมือหลักสำหรับติดตั้งไลบรารีจากบุคคลที่สามคือ `pip` ซึ่งเป็นตัวจัดการแพ็กเกจของ Python โดยสามารถใช้คำสั่งง่าย ๆ เช่น `pip install` เพื่อติดตั้งไลบรารีใหม่เข้าสู่สภาพแวดล้อมของ Python ได้อย่างรวดเร็ว ความสะดวกนี้ช่วยส่งเสริมให้เกิดการพัฒนาแอปพลิเคชันที่แข็งแกร่งได้อย่างรวดเร็ว เพราะสามารถนำประโยชน์จากงานพัฒนาของชุมชน Python มาใช้ได้ ไม่ว่าจะเป็นการคำนวณเชิงตัวเลข การวิเคราะห์ข้อมูล การพัฒนาเว็บ หรือการเรียนรู้ของเครื่อง ก็มีไลบรารีให้เลือกใช้หลากหลาย ส่งเสริมให้เกิดนวัตกรรมและประสิทธิภาพในการพัฒนาโปรแกรม

10.6 การติดตั้งไลบรารีจากบุคคลที่สาม

คุณสามารถติดตั้งไลบรารีจากบุคคลที่สามได้โดยใช้คำสั่ง `pip install` คำสั่งนี้จะช่วยให้คุณสามารถดาวน์โหลดและติดตั้งไลบรารีจาก PyPI ได้อย่างง่ายดาย เพียงเปิดโปรแกรมบรรทัดคำสั่ง (Command Line Interface) และพิมพ์ `pip install library_name` เพื่อเพิ่มไลบรารีที่ต้องการเข้าสู่สภาพแวดล้อมของ Python

```
pip install requests
```

10.7 ตัวอย่างไลบรารียอดนิยมจากบุคคลที่สาม

ไลบรารีจากบุคคลที่สามที่ได้รับความนิยมใน Python ช่วยเพิ่มความสามารถให้กับภาษาได้อย่างมาก และถูกใช้อย่างแพร่หลายในหลากหลายสาขา ไลบรารีเหล่านี้ได้แก่ `NumPy` สำหรับการคำนวณเชิงตัวเลข `Pandas` สำหรับ

การจัดการและวิเคราะห์ข้อมูล **Matplotlib** สำหรับการแสดงผลข้อมูล **Requests** สำหรับจัดการคำขอ HTTP และ **Flask** สำหรับการพัฒนาเว็บ

ไลบรารีแต่ละตัวให้เครื่องมือเฉพาะทางที่ช่วยให้การจัดการงานซับซ้อนเป็นไปอย่างง่ายดายและมีประสิทธิภาพ สามารถเข้าถึงได้ง่ายจาก PyPI และติดตั้งได้ด้วยคำสั่ง **pip** การใช้ไลบรารีเหล่านี้ในโปรเจกต์ของคุณจะช่วยให้คุณสามารถใช้ประโยชน์จากผลงานของชุมชน Python และพัฒนาแอปพลิเคชันได้อย่างมีพลังและมีประสิทธิภาพมากขึ้น

10.7.1 NumPy

NumPy เป็นไลบรารีที่ทรงพลังสำหรับการคำนวณเชิงตัวเลขในภาษา Python ซึ่งถูกใช้กันอย่างแพร่หลายในด้านการคำนวณทางวิทยาศาสตร์ การวิเคราะห์ข้อมูล และการเรียนรู้ของเครื่อง ไลบรารีนี้ให้การสนับสนุนอย่างเต็มที่สำหรับอาเรย์และเมทริกซ์ ซึ่งเป็นโครงสร้างข้อมูลพื้นฐานที่จำเป็นสำหรับการคำนวณเชิงตัวเลขอย่างมีประสิทธิภาพ

NumPy ประกอบด้วยฟังก์ชันทางคณิตศาสตร์ที่หลากหลาย เช่น พีชคณิตเชิงเส้น การวิเคราะห์ทางสถิติ และการแปลงฟูริเยร์ ออบเจกต์อาเรย์ของ NumPy ได้รับการปรับแต่งให้มีประสิทธิภาพสูง จึงสามารถดำเนินการคำนวณได้อย่างรวดเร็ว นอกจากนี้ยังสามารถรวมการใช้งานร่วมกับไลบรารีวิทยาศาสตร์อื่น ๆ เช่น **SciPy** และ **Pandas** เพื่อสร้างเวิร์กโฟลว์สำหรับการจัดการและวิเคราะห์ข้อมูลที่ซับซ้อนได้อย่างราบรื่น ด้วยความสามารถในการจัดการข้อมูลขนาดใหญ่และการดำเนินการทางคณิตศาสตร์ขั้นสูง **NumPy** จึงเป็นเครื่องมือพื้นฐานที่นักพัฒนาและนักวิจัยในสายงานที่ต้องการการประมวลผลเชิงตัวเลขไม่ควรขาด

```python
import numpy as np

# Create a 3x3 array of random integers between 1 and 10
random_matrix = np.random.randint(1, 11, size=(3, 3))
print("Random 3x3 Matrix:\n", random_matrix)

# Find the sum of all elements in the matrix
matrix_sum = np.sum(random_matrix)
print(f"\nSum of all elements: {matrix_sum}")

# Find the mean of the matrix
matrix_mean = np.mean(random_matrix)
print(f"\nMean of the matrix: {matrix_mean:2.2f}")

# Transpose the matrix
transposed_matrix = np.transpose(random_matrix)
print("\nTransposed Matrix:\n", transposed_matrix)
```

Listing 10.10: Example of NumPy

10.7.2 Pandas

Pandas เป็นไลบรารีที่มีความยืดหยุ่นและทรงพลังสำหรับการจัดการและวิเคราะห์ข้อมูลในภาษา Python ซึ่งได้รับความนิยมในสาขาวิทยาศาสตร์ข้อมูล การเงิน เศรษฐศาสตร์ และสาขาอื่น ๆ ที่เกี่ยวข้องกับชุดข้อมูลขนาดใหญ่ โดยไลบรารีนี้ให้โครงสร้างข้อมูลที่มีประสิทธิภาพ เช่น **Series** และ **DataFrame** ซึ่งเหมาะสำหรับการจัดการข้อมูลแบบตาราง

DataFrame โดยเฉพาะ เป็นโครงสร้างข้อมูลแบบสองมิติ ที่สามารถปรับขนาดได้ และเก็บข้อมูลหลากหลายประเภท พร้อมทั้งมีป้ายกำกับแถวและคอลัมน์ คล้ายกับสเปรดชีตหรือฐานข้อมูล SQL **Pandas** มีฟังก์ชันครบถ้วนสำหรับการจัดการข้อมูล เช่น การอ่าน/เขียนจากหลายรูปแบบไฟล์ (CSV, Excel, SQL) การทำความสะอาดข้อมูล การกรอง การรวมข้อมูล การจัดกลุ่ม และการปรับโครงสร้างข้อมูล

นอกจากนี้ **Pandas** ยังสามารถทำงานร่วมกับไลบรารีอื่น เช่น **NumPy** และ **Matplotlib** เพื่อสร้างเวิร์กโฟลว์สำหรับการวิเคราะห์และแสดงผลข้อมูลได้อย่างราบรื่น ด้วยเครื่องมือที่เข้าใจง่ายแต่ทรงพลัง **Pandas** ทำให้กระบวนการวิเคราะห์ข้อมูลง่ายขึ้นทั้งสำหรับผู้เริ่มต้นและผู้เชี่ยวชาญ ช่วยให้สามารถสกัดข้อมูลเชิงลึกจากชุดข้อมูลที่ซับซ้อนได้อย่างมีประสิทธิภาพ

```python
import pandas as pd

# Create a DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Los Angeles', 'Chicago']}

df = pd.DataFrame(data)
print("DataFrame:\n", df)

# Calculate the average age
average_age = df['Age'].mean()
print("\nAverage Age:", average_age)

# Filter rows where Age is greater than 28
filtered_df = df[df['Age'] > 28]
print("\nFiltered DataFrame (Age > 28):\n", filtered_df)

# Add a new column for salary
df['Salary'] = [50000, 60000, 70000]
print("\nDataFrame with Salary column:\n", df)
```

Listing 10.11: Example of Pandas

10.7.3 Matplotlib

Matplotlib เป็นไลบรารีที่มีความยืดหยุ่นสูงสำหรับการสร้างภาพข้อมูลทั้งแบบคงที่ แบบเคลื่อนไหว และแบบอินเทอร์แอคทีฟในภาษา Python ได้รับความนิยมอย่างกว้างขวางในด้านวิทยาศาสตร์ข้อมูล วิศวกรรม การเงิน และสาขาอื่น ๆ ที่ต้องการการแสดงผลข้อมูลที่ชัดเจนและเข้าใจง่าย

ด้วย **Matplotlib** คุณสามารถสร้างกราฟและแผนภูมิได้หลากหลายรูปแบบ เช่น กราฟเส้น แผนภูมิแท่ง ฮิสโตแกรม กราฟกระจาย และกราฟสามมิติ การออกแบบของไลบรารีนี้มีลักษณะคล้ายกับ MATLAB จึงเหมาะสำหรับผู้ใช้ที่คุ้นเคยกับ MATLAB อยู่แล้ว **Matplotlib** ให้คุณควบคุมองค์ประกอบของกราฟได้ละเอียด เช่น สี ป้ายชื่อ สไตล์เส้น และฟอนต์ ทำให้สามารถปรับแต่งกราฟให้เหมาะกับความต้องการเฉพาะได้

นอกจากนี้ยังสามารถทำงานร่วมกับไลบรารีวิทยาศาสตร์อื่น ๆ เช่น **NumPy** และ **Pandas** ได้อย่างไร้รอยต่อ ทำให้ **Matplotlib** เป็นเครื่องมือสำคัญในการวิเคราะห์และนำเสนอข้อมูล อีกทั้งยังรองรับการสร้างกราฟแบบอินเทอร์แอคทีฟ ให้ผู้ใช้สามารถสำรวจข้อมูลผ่านการซูม เลื่อน และอัปเดตกราฟแบบไดนามิกได้ ความยืดหยุ่นและใช้งานง่ายของ **Matplotlib** ทำให้เป็นเครื่องมือที่ทรงพลังในการแปลงข้อมูลให้เป็นกราฟิกที่น่าสนใจและสื่อความหมายได้อย่างมีประสิทธิภาพ

```python
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

# Create a line plot
plt.plot(x, y, marker='o', linestyle='-', color='b')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')

# Display the plot
plt.show()
```

Listing 10.12: Example of Matplotlib

```python
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.animation as animation

# Create a figure and an axis
fig, ax = plt.subplots()

# Generate initial data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Create a line object that will be updated during the animation
line, = ax.plot(x, y)

# Function to update the plot for each frame
def update(frame):
    line.set_ydata(np.sin(x + frame / 10.0))  # Shift the sine wave
    return line,

# Create an animation object
ani = animation.FuncAnimation(fig, update, frames=100, interval=50, blit=
    True)

# Display the animation
plt.show()
```

Listing 10.13: Example of Animation Matplotlib

10.7.4 Requests

Requests เป็นไลบรารียอดนิยมที่ใช้งานง่ายสำหรับการส่งคำขอ HTTP ในภาษา Python มันช่วยให้การโต้ตอบกับเว็บเซอร์วิสและ API เป็นเรื่องง่าย โดยนักพัฒนาสามารถส่งคำขอและจัดการการตอบกลับได้อย่างไม่ซับซ้อน

ด้วยอินเทอร์เฟซที่เรียบง่ายของ Requests คุณสามารถใช้เมธอด HTTP ได้หลากหลาย เช่น **GET**, **POST**, **PUT**, **DELETE** และอื่น ๆ นอกจากนี้ยังสามารถจัดการเรื่องที่ซับซ้อน เช่น การยืนยันตัวตน เซสชัน คุกกี้ และการเปลี่ยนเส้นทาง ได้อย่างมีประสิทธิภาพ

Requests เหมาะสำหรับงานเชื่อมต่อ API, ดึงข้อมูลจากเว็บไซต์ (web scraping) และการสื่อสารผ่านเครือข่าย โดยมีไวยากรณ์ที่เข้าใจง่ายและเอกสารประกอบที่ครอบคลุม จึงถือเป็นเครื่องมือสำคัญสำหรับนักพัฒนาที่ต้องติดต่อกับเว็บผ่านแอปพลิเคชัน Python

```python
import requests

# Make a GET request to a public API (GitHub API)
response = requests.get('https://api.github.com/users/octocat')

# Check if the request was successful
if response.status_code == 200:
    user_data = response.json()

    # Display some information from the response
    print(f"Username: {user_data['login']}")
    print(f"Name: {user_data['name']}")
    print(f"Bio: {user_data['bio']}")
    print(f"Public Repos: {user_data['public_repos']}")
    print(f"Followers: {user_data['followers']}")
    print(f"Following: {user_data['following']}")
else:
    print("Failed to retrieve data.")
```

Listing 10.14: Example of Requests

10.7.5 Flask

Flask เป็นไมโครเฟรมเวิร์กที่เบาและยืดหยุ่นสำหรับการพัฒนาเว็บแอปพลิเคชันด้วยภาษา Python ออกแบบมาให้ใช้งานง่าย เหมาะทั้งสำหรับผู้เริ่มต้นและนักพัฒนาที่มีประสบการณ์ โดยมีเครื่องมือพื้นฐานในการสร้างเว็บ เช่น การกำหนดเส้นทาง การจัดการคำขอ และการแสดงผลด้วยเทมเพลต

Flask เน้นความเรียบง่าย โดยไม่กำหนดโครงสร้างตายตัวให้กับแอปพลิเคชัน ผู้พัฒนาสามารถจัดโครงสร้างแอปตามที่ต้องการได้อย่างอิสระ นอกจากนี้ยังสามารถเพิ่มฟีเจอร์เพิ่มเติมได้ผ่านส่วนขยาย

Flask ทำงานบนมาตรฐาน WSGI (Web Server Gateway Interface) ซึ่งทำให้สามารถใช้งานร่วมกับเว็บเซิร์ฟเวอร์ต่าง ๆ ได้ง่าย และสามารถรวมเข้ากับไลบรารีอื่นของ Python ได้อย่างดี เช่น **SQLAlchemy** สำหรับจัดการฐานข้อมูล และ **Jinja2** สำหรับระบบเทมเพลต ความเรียบง่ายและความยืดหยุ่นของ **Flask** ทำให้มันเป็นตัวเลือกยอดนิยมสำหรับการพัฒนาเว็บแอป ตั้งแต่โปรเจกต์ต้นแบบไปจนถึงระบบขนาดใหญ่

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Flask!"

if __name__ == '__main__':
    app.run(debug=True)
```

Listing 10.15: Example of Flask

```python
from flask import Flask, render_template_string

app = Flask(__name__)

# HTML template as a string
html_template = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Greeting Page</title>
</head>
<body>
    <h1>Hello, {{ name }}!</h1>
    <p>Welcome to your simple Flask web app.</p>
</body>
</html>
"""

# Define a route for the homepage
@app.route('/')
def home():
    return render_template_string(html_template, name="Alice")

# Define a dynamic route that takes a name parameter
@app.route('/greet/<name>')
def greet(name):
    return render_template_string(html_template, name=name)

if __name__ == '__main__':
    app.run(debug=True)
```

Listing 10.16: Example of Flask II

10.7.6 SQLite

`SQLite` เป็นระบบจัดการฐานข้อมูลเชิงสัมพันธ์ (Relational Database Management System: RDBMS) ที่มีน้ำหนักเบา ไม่ต้องใช้เซิร์ฟเวอร์ และเป็นแบบ self-contained ซึ่งได้รับความนิยมอย่างแพร่หลายในแอปพลิเคชันหลากหลาย เนื่องจากใช้งานง่ายและไม่ต้องมีการกำหนดค่าที่ซับซ้อน

`SQLite` เป็นโอเพ่นซอร์สและฝังตัวอยู่ภายในแอปพลิเคชันโดยตรง แตกต่างจากฐานข้อมูลแบบดั้งเดิมอย่าง `MySQL` หรือ `PostgreSQL` ที่ต้องพึ่งพาเซิร์ฟเวอร์ภายนอก SQLite ทำงานภายในกระบวนการของแอปพลิเคชันโดยตรง ทำให้เหมาะสำหรับโครงการขนาดเล็ก ระบบฝังตัว (embedded systems) อุปกรณ์พกพา และแอปพลิเคชันที่ไม่ต้องการความซับซ้อนของระบบฐานข้อมูลแบบเต็มรูปแบบ

```python
import sqlite3

# Connect to SQLite (creates a database file if it doesn't exist)
conn = sqlite3.connect('mydatabase.db')

# Create a cursor object to interact with the database
cur = conn.cursor()

# Create a table
cur.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        age INTEGER NOT NULL,
        city TEXT NOT NULL
    )
''')

# Insert some data into the table
cur.execute("INSERT INTO users (name, age, city) VALUES ('Alice', 25, 'New York')")
cur.execute("INSERT INTO users (name, age, city) VALUES ('Bob', 30, 'Los Angeles')")
cur.execute("INSERT INTO users (name, age, city) VALUES ('Charlie', 35, 'Chicago')")

# Commit the changes
conn.commit()

# Query the data and display the results
cur.execute("SELECT * FROM users WHERE age > 28")
rows = cur.fetchall()

print("Users older than 28:")
for row in rows:
    print(f"ID: {row[0]}, Name: {row[1]}, Age: {row[2]}, City: {row[3]}")

# Close the connection
conn.close()
```

Listing 10.17: Example of SQLite

10.8 ตัวอย่างการใช้งานจริง

บทนี้นำเสนอชุดตัวอย่างการใช้งานจริง เพื่อแสดงให้เห็นว่าคุณสามารถผสมผสานการใช้โมดูลและไลบรารีต่าง ๆ เพื่อดำเนินงานที่ซับซ้อนได้อย่างไร ตัวอย่างเช่น การดึงข้อมูลจาก Web API โดยใช้ไลบรารี `requests` จากนั้นประมวลผลและวิเคราะห์ข้อมูลด้วย `pandas` และแสดงผลด้วย `matplotlib`

ตัวอย่างเหล่านี้ช่วยเสริมความเข้าใจในแนวคิดที่ได้นำเสนอในบท โดยเน้นให้เห็นถึงการใช้งานเครื่องมือและเทคนิคต่าง ๆ ในสถานการณ์จริง เมื่อผู้อ่านลงมือทำตามตัวอย่างเหล่านี้ จะสามารถเข้าใจการทำงานร่วมกันของไลบรารีต่าง ๆ ในการสร้างแอปพลิเคชัน Python ที่มีประสิทธิภาพ ทรงพลัง และยืดหยุ่นมากยิ่งขึ้น

10.8.1 ตัวอย่าง: การใช้โมดูลหลายตัวร่วมกัน

การรวมโมดูลหลายตัวเพื่อดำเนินการที่ซับซ้อน เป็นการใช้ประโยชน์จากจุดแข็งของไลบรารีแต่ละตัวเพื่อสร้างทางออกที่ครบถ้วน ตัวอย่างเช่น คุณอาจใช้โมดูล `os` สำหรับจัดการระบบไฟล์ โมดูล `json` สำหรับแปลงและจัดการข้อมูล JSON และไลบรารี `requests` สำหรับเชื่อมต่อกับ Web API

เมื่อรวมโมดูลเหล่านี้เข้าด้วยกัน คุณสามารถสร้างสคริปต์ที่ดึงข้อมูลจากแหล่งข้อมูลออนไลน์ ประมวลผล และจัดเก็บข้อมูลไว้ในรูปแบบที่มีโครงสร้างอย่างเป็นระบบ วิธีการนี้แสดงให้เห็นถึงพลังของการออกแบบแบบแยกส่วนของ Python ที่ช่วยให้นักพัฒนาสามารถสร้างแอปพลิเคชันที่ซับซ้อนได้อย่างมีประสิทธิภาพ โดยใช้เครื่องมือที่มีอยู่ซ้ำและรวมเข้าด้วยกัน

```python
import os
import json
import requests

def fetch_github_user(username):
    url = f"https://api.github.com/users/{username}"
    response = requests.get(url)
    return response.json()

def save_user_data(username):
    user_data = fetch_github_user(username)
    with open(f"{username}.json", "w") as file:
        json.dump(user_data, file)

def main():
    username = input("Enter GitHub username: ")
    save_user_data(username)
    print(f"User data saved to {username}.json")

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

Listing 10.18: Example of Using Multiple Modules

บทที่ 10 โจทย์และแบบฝึกหัด: ไลบรารีและโมดูลใน Python

10.1 สร้างโมดูลและเรียกใช้ฟังก์ชัน
สร้างโมดูลชื่อ `mymodule.py` *ซึ่งมีฟังก์ชันชื่อ* `greet(name)` *ที่แสดงข้อความทักทาย แล้วเขียนสคริปต์เพื่อนำเข้าโมดูลและเรียกใช้ฟังก์ชันนี้*

10.2 นำเข้าฟังก์ชันเฉพาะจากโมดูล
นำเข้า `sqrt` *และ* `pi` *จากโมดูล* `math` *และคำนวณค่ารากที่สองของ* `pi`

10.3 ใช้นามแฝงในการนำเข้าโมดูล
นำเข้าโมดูล `datetime` *โดยใช้นามแฝงว่า* `dt` *และแสดงวันที่และเวลาปัจจุบัน*

10.4 นำเข้าทุกอย่างจากโมดูล
นำเข้าทุกฟังก์ชันจากโมดูล `math` *และคำนวณค่า sine ของ* `pi/2` *พร้อมอธิบายว่าทำไมแนวทางนี้จึงไม่แนะนำให้ใช้*

10.5 สร้างและเปลี่ยนไดเรกทอรีด้วย os
ใช้โมดูล `os` *เพื่อสร้างไดเรกทอรีใหม่ชื่อ* `test_dir` *และเปลี่ยนไดเรกทอรีปัจจุบันไปยัง* `test_dir`

10.6 แสดงเวอร์ชันของ Python
ใช้โมดูล `sys` *เพื่อแสดงเวอร์ชันของ Python ที่ใช้อยู่ในระบบ*

10.7 สร้างออบเจกต์วันที่
ใช้โมดูล `datetime` *เพื่อสร้างออบเจกต์วันที่สำหรับวันที่ 1 มกราคม 2025 และแสดงผลลัพธ์*

10.8 สุ่มเลขทศนิยมระหว่าง 0 ถึง 1
ใช้โมดูล `random` *เพื่อสุ่มเลขทศนิยมแบบลอยตัวระหว่าง 0 ถึง 1*

10.9 แปลง Dictionary เป็น JSON และกลับมาเป็น Dictionary
ใช้โมดูล `json` *เพื่อแปลง Dictionary* `{"name": "Bob", "age": 30}` *ให้เป็น JSON string แล้วแปลงกลับมาเป็น Dictionary อีกครั้ง*

10.10 ติดตั้งและใช้ไลบรารี requests
ติดตั้งไลบรารี `requests` *ด้วยคำสั่ง* `pip` *และใช้เพื่อส่งคำขอ GET ไปยัง* https://api.github.com *แล้วแสดงรหัสสถานะของการตอบกลับ*

10.11 สร้างอาเรย์ด้วย NumPy
ใช้ไลบรารี `numpy` *เพื่อสร้างอาเรย์ขนาด 2x2 ที่มีค่า* `[[1, 2], [3, 4]]` *และแสดงผลลัพธ์*

10.12 ดึงและบันทึกข้อมูลผู้ใช้จาก GitHub API
ใช้โมดูล `os`, `json`, *และ* `requests` *เพื่อสร้างสคริปต์ที่ดึงข้อมูลผู้ใช้จาก GitHub API และบันทึกไว้ในไฟล์* `user.json` *ภายในไดเรกทอรีใหม่ชื่อ* `github_data`

ภาคผนวก

ภาคผนวกนี้จัดเตรียมแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งานจริง เพื่อเสริมความเข้าใจของคุณเกี่ยวกับเนื้อหาในบทที่ 10

A10.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Python for Data Analysis* โดย Wes McKinney
- *Fluent Python* โดย Luciano Ramalho

บทเรียนออนไลน์:

- Python Modules - W3Schools
- An Introduction to Python's Standard Library - Real Python

คอร์สออนไลน์:

- Coursera: Applied Data Science with Python โดย University of Michigan
- Udacity: Intermediate Python

A10.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). Python Documentation - Modules. เรียกดูจาก https://docs.python.org/3/tutorial/modules.html
- Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook* (ฉบับที่ 3). สำนักพิมพ์ O'Reilly Media.

A10.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การใช้งานโมดูลพื้นฐาน

- เขียนโปรแกรม Python ที่ใช้โมดูล `math` เพื่อคำนวณรากที่สองของตัวเลข
- สร้างสคริปต์ที่นำเข้าโมดูล `datetime` และแสดงวันที่และเวลาปัจจุบัน

แบบฝึกหัดที่ 2: โมดูลที่สร้างเอง

- เขียนโมดูล Python ชื่อ `mymath` ซึ่งมีฟังก์ชันสำหรับ บวก ลบ คูณ และหาร จากนั้นนำมาใช้งานในสคริปต์อื่น
- สร้างโมดูลชื่อ `greetings` ที่มีฟังก์ชัน `say_hello` เพื่อแสดงข้อความทักทาย และใช้งานโมดูลนี้ในสคริปต์เพื่อทักทายผู้ใช้

แบบฝึกหัดที่ 3: การใช้ไลบรารีจากภายนอก

- เขียนสคริปต์ Python ที่ใช้ไลบรารี `requests` เพื่อดึงข้อมูลจาก Web API และแสดงผล

- พัฒนาโปรแกรมที่ใช้ไลบรารี `pandas` เพื่ออ่านไฟล์ CSV และแสดงเนื้อหาภายใน

แบบฝึกหัดที่ 4: การสร้างและใช้งานแพ็กเกจ

- สร้างแพ็กเกจชื่อ `utilities` ที่มีสองโมดูลคือ `file_utils` สำหรับการจัดการไฟล์ และ `string_utils` สำหรับการจัดการสตริง แล้วใช้งานในสคริปต์
- เขียนโปรแกรมที่ใช้ไลบรารี `matplotlib` เพื่อสร้างกราฟอย่างง่ายจากชุดข้อมูลที่กำหนด

A10.4 ตัวอย่างการใช้งานจริง

ตัวอย่างที่ 1: การใช้โมดูล `math`

```
import math

# Calculate square root
number = 16
sqrt = math.sqrt(number)
print(f"The square root of {number} is {sqrt}")
```

Listing 10.19: Using the math module

ตัวอย่างที่ 2: การใช้โมดูล `datetime`

```
import datetime

# Get current date and time
current_datetime = datetime.datetime.now()
print("Current date and time:", current_datetime)
```

Listing 10.20: Using the datetime module

ตัวอย่างที่ 3: โมดูลที่สร้างเอง `mymath`

```
# mymath.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Division by zero is not allowed."
    return a / b
```

Listing 10.21: Creating mymath module

การใช้งานโมดูล mymath

```python
# main.py
import mymath

print(mymath.add(5, 3))         # Output: 8
print(mymath.subtract(5, 3))    # Output: 2
print(mymath.multiply(5, 3))    # Output: 15
print(mymath.divide(5, 0))      # Output: Division by zero is not allowed.
```

Listing 10.22: Using mymath module

ตัวอย่างที่ 4: การใช้ไลบรารี requests

```python
import requests

# Fetch data from a web API
response = requests.get('https://api.github.com')
print(response.json())
```

Listing 10.23: Using the requests library

ตัวอย่างที่ 5: การใช้ไลบรารี pandas

```python
import pandas as pd

# Read a CSV file
df = pd.read_csv('data.csv')
print(df.head())
```

Listing 10.24: Using the pandas library

ตัวอย่างที่ 6: การสร้างและใช้งานแพ็กเกจ

```
# utilities/
#     __init__.py
#     file_utils.py
#     string_utils.py

# file_utils.py
def read_file(file_path):
    with open(file_path, 'r') as file:
        return file.read()

def write_file(file_path, content):
    with open(file_path, 'w') as file:
        file.write(content)

# string_utils.py
def to_uppercase(s):
    return s.upper()

def to_lowercase(s):
    return s.lower()
```

Listing 10.25: Creating package structure

การใช้งานแพ็กเกจ `utilities`

```
# main.py
from utilities import file_utils, string_utils

file_content = file_utils.read_file('example.txt')
print(string_utils.to_uppercase(file_content))
```

Listing 10.26: Using the utilities package

ตัวอย่างที่ 7: การใช้ไลบรารี `matplotlib`

```
import matplotlib.pyplot as plt

# Plot a simple graph
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Simple Plot')
plt.show()
```

Listing 10.27: Using the matplotlib library

บทที่ 11

การเขียนโปรแกรมเชิงวัตถุ

บทนี้นำเสนอภาพรวมอย่างครบถ้วนเกี่ยวกับการเขียนโปรแกรมเชิงวัตถุ (OOP) ในภาษา Python โดยอธิบายแนวคิดพื้นฐานต่าง ๆ อย่างละเอียด เช่น คลาส (class), อ็อบเจกต์ (object), การสืบทอด (inheritance), การห่อหุ้ม (encapsulation) และพอลิมอร์ฟิซึม (polymorphism) โดยเริ่มต้นจากการแนะนำแนวคิด OOP และหลักการสำคัญต่าง ๆ จากนั้นกล่าวถึงคลาสและอ็อบเจกต์ซึ่งอธิบายถึงการใช้คลาสเป็นแบบพิมพ์เขียวสำหรับสร้างอ็อบเจกต์ พร้อมตัวอย่างประกอบ

ในส่วนของการสืบทอดจะแสดงให้เห็นว่าคลาสสามารถรับคุณสมบัติและเมธอดจากคลาสแม่ได้อย่างไร เพื่อเพิ่มความสามารถในการนำโค้ดกลับมาใช้ซ้ำและทำให้โครงสร้างโปรแกรมเป็นระบบมากขึ้น ถัดมาเป็นการห่อหุ้ม ซึ่งเน้นการจำกัดการเข้าถึงข้อมูลภายในอ็อบเจกต์เพื่อป้องกันการเปลี่ยนแปลงโดยไม่ตั้งใจ พร้อมด้วยตัวอย่างประกอบ

พอลิมอร์ฟิซึมถูกอธิบายผ่านตัวอย่างของการเขียนเมธอดซ้ำ (method overriding) และฟังก์ชันที่ทำงานแตกต่างกันตามอ็อบเจกต์ที่ใช้งาน เพื่อแสดงถึงความยืดหยุ่นในการออกแบบโปรแกรม บทนี้ยังรวมถึงตัวอย่างการใช้งานจริง เช่น การสร้างคลาสบัญชีธนาคารแบบง่าย และลำดับชั้นของรูปร่างเรขาคณิตที่มีการคำนวณพื้นที่ เพื่อเสริมความเข้าใจแนวคิด OOP ผ่านสถานการณ์ที่ใกล้เคียงกับความเป็นจริง

โดยรวม บทนี้มีจุดประสงค์เพื่อเสริมสร้างพื้นฐานที่แข็งแกร่งด้าน OOP ให้กับผู้อ่าน เพื่อสามารถเขียนโค้ดที่เป็นโมดูล นำกลับมาใช้ได้ และดูแลรักษาง่ายในภาษา Python

11.1 บทนำสู่การเขียนโปรแกรมเชิงวัตถุ

การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) เป็นแนวคิดการเขียนโปรแกรมที่ทรงพลัง โดยมี "อ็อบเจกต์" เป็นแกนหลัก ซึ่งอ็อบเจกต์คือตัวอย่าง (instance) ของคลาส ที่รวมข้อมูลไว้ในรูปของแอตทริบิวต์และพฤติกรรมในรูปของเมธอด OOP ถูกออกแบบมาเพื่อช่วยจัดระเบียบโค้ด เพิ่มความสามารถในการนำกลับมาใช้และขยายระบบได้อย่างมีประสิทธิภาพ

หลักการสำคัญของ OOP ได้แก่ การห่อหุ้ม (encapsulation) ซึ่งเป็นการรวมข้อมูลและเมธอดไว้ในหน่วยเดียว การสืบทอด (inheritance) ซึ่งทำให้สามารถสร้างคลาสใหม่โดยใช้คุณสมบัติของคลาสเดิม และพอลิมอร์ฟิซึม (polymorphism) ซึ่งทำให้เมธอดสามารถทำงานต่างกันตามอ็อบเจกต์ที่ใช้งาน

OOP มอบโครงสร้างที่ชัดเจนโดยการจัดโปรแกรมให้เป็นอ็อบเจกต์และคลาส ช่วยให้จัดการระบบที่ซับซ้อนได้ง่ายขึ้น และเชื่อมโยงแนวคิดในโลกจริงเข้ากับโค้ดได้อย่างเป็นธรรมชาติ บทนี้จะเจาะลึกแนวคิดเหล่านี้พร้อมตัวอย่างการใช้งานใน Python

การเขียนโปรแกรมเชิงวัตถุ (OOP) คือแนวคิดที่เน้นการใช้คลาสและอ็อบเจกต์ โดยแนวคิดสำคัญประกอบด้วย:

การเขียนโปรแกรมเชิงวัตถุ (OOP)

- **อ็อบเจกต์ (Objects)**: อินสแตนซ์ของคลาสที่มีข้อมูลและเมธอดในตัว
- **คลาส (Classes)**: แม่แบบสำหรับสร้างอ็อบเจกต์ โดยกำหนดโครงสร้างและพฤติกรรม
- **การห่อหุ้ม (Encapsulation)**: รวมข้อมูลและเมธอดในหน่วยเดียว และจำกัดการเข้าถึง
- **การสืบทอด (Inheritance)**: คลาสใหม่สามารถสืบทอดคุณสมบัติและเมธอดจากคลาสเดิม
- **พอลิมอร์ฟิซึม (Polymorphism)**: เมธอดสามารถทำงานต่างกันตามอ็อบเจกต์ที่ใช้งาน
- **นามธรรม (Abstraction)**: ซ่อนรายละเอียดที่ซับซ้อน และเปิดเผยเฉพาะสิ่งจำเป็นให้ผู้ใช้เห็น
- **ความเป็นโมดูล (Modularity)**: แยกโค้ดเป็นส่วน ๆ ช่วยให้เข้าใจและจัดการได้ง่าย
- **การนำกลับมาใช้ (Reusability)**: เขียนโค้ดให้สามารถนำไปใช้ซ้ำได้ในหลายส่วนหรืองานอื่น

การเขียนโปรแกรมเชิงวัตถุ (OOP) และการเขียนโปรแกรมเชิงโพรซีเยอร์ (Procedural-Oriented Programming: POP) เป็นสองแนวคิดหลักในการเขียนโปรแกรม โดยมีวิธีจัดการโครงสร้างและการทำงานของโค้ดแตกต่างกัน

OOP เน้นการใช้คลาสและอ็อบเจกต์ โดยอ็อบเจกต์จะมีคุณสมบัติ (แอตทริบิวต์) และพฤติกรรม (เมธอด) ที่กำหนดไว้ในคลาส แนวคิดนี้เน้นความเป็นโมดูล การนำกลับมาใช้ซ้ำ และการดูแลรักษาโค้ด โดยใช้หลักการสำคัญ เช่น การห่อหุ้ม การสืบทอด และพอลิมอร์ฟิซึม ซึ่งช่วยให้โค้ดมีโครงสร้างดี ยืดหยุ่น และขยายได้ง่าย OOP ใช้แนวทางจากล่างขึ้นบน (bottom-up) ซึ่งเหมาะสำหรับซอฟต์แวร์ที่ซับซ้อนและมีขนาดใหญ่ เช่น Python, Java, C++, และ Ruby

ในทางตรงกันข้าม POP มุ่งเน้นไปที่การทำงานเป็นลำดับ โดยแบ่งโค้ดเป็นฟังก์ชันต่าง ๆ ที่ทำงานตามขั้นตอน ซึ่งเหมาะกับโปรแกรมขนาดเล็กที่ไม่มีความซับซ้อนสูง POP ไม่มีแนวคิดการห่อหุ้ม ทำให้ข้อมูลสามารถเข้าถึงได้จากทุกฟังก์ชัน ซึ่งส่งผลต่อความปลอดภัยของข้อมูล และทำให้การดูแลรักษาโค้ดยากขึ้น ภาษาในกลุ่มนี้ได้แก่ C, Pascal, และ FORTRAN

โดยสรุป OOP เหมาะกับระบบที่ซับซ้อนและต้องการการขยายในระยะยาว ในขณะที่ POP เหมาะสำหรับงานที่ง่ายและต้องการโครงสร้างที่ตรงไปตรงมา

คุณสมบัติ	OOP (การเขียนโปรแกรมเชิงวัตถุ)	POP (การเขียนโปรแกรมเชิงโพรซีเยอร์)
แนวคิดหลัก	อิงกับอ็อบเจกต์และคลาส	อิงกับฟังก์ชันและกระบวนการ
จุดเน้นหลัก	เน้นอ็อบเจกต์ที่รวมข้อมูลและเมธอดไว้ด้วยกัน	เน้นลำดับขั้นตอนการทำงาน (procedures)
การเข้าถึงข้อมูล	จำกัดการเข้าถึงผ่านเมธอด (encapsulation)	ฟังก์ชันสามารถเข้าถึงและเปลี่ยนข้อมูลได้โดยตรง
ความเป็นโมดูล	โค้ดแบ่งเป็นอ็อบเจกต์ (อินสแตนซ์ของคลาส)	โค้ดแบ่งเป็นฟังก์ชันหรือขั้นตอน
การนำกลับมาใช้	มีสูง: ผ่านการสืบทอด พอลิมอร์ฟิซึม และคลาส	มีน้อย: มักต้องคัดลอกโค้ดซ้ำ
ความปลอดภัยของข้อมูล	สูง: มีการป้องกันข้อมูลด้วยการห่อหุ้ม	ต่ำ: ข้อมูลเปิดให้ทุกฟังก์ชันเข้าถึงได้
การดูแลรักษาโค้ด	ง่าย: โครงสร้างเป็นโมดูล ปรับปรุงได้เป็นส่วน ๆ	ยาก: การเปลี่ยนแปลงอาจกระทบหลายส่วน
ภาษาโปรแกรมตัวอย่าง	Python, Java, C++, Ruby, C#	C, Pascal, FORTRAN, Basic
การทำงาน	เน้นการโต้ตอบระหว่างอ็อบเจกต์	ทำงานเป็นลำดับจากบนลงล่าง
การสืบทอด	รองรับ: คลาสสามารถสืบทอดกันได้	ไม่รองรับการสืบทอด
พอลิมอร์ฟิซึม	รองรับ: เมธอดเดียวกันใช้กับอ็อบเจกต์ต่างกันได้	ไม่รองรับพอลิมอร์ฟิซึม
นามธรรม	รองรับ: ซ่อนความซับซ้อนจากผู้ใช้	ไม่มีแนวคิดเรื่องนามธรรม
แนวทางการพัฒนา	จากล่างขึ้นบน (Bottom-up)	จากบนลงล่าง (Top-down)

Table 11.1: เปรียบเทียบระหว่างการเขียนโปรแกรมเชิงวัตถุ (OOP) และเชิงโพรซีเยอร์ (POP)

11.2 คลาสและอ็อบเจกต์

คลาสและอ็อบเจกต์เป็นพื้นฐานสำคัญของการเขียนโปรแกรมเชิงวัตถุ (OOP) การเข้าใจวิธีใช้อย่างมีประสิทธิภาพมีความสำคัญอย่างยิ่งในการเขียนโค้ดให้สะอาด เป็นโมดูล และนำกลับมาใช้ได้

คลาส (Classes)

คลาสคือแบบพิมพ์เขียวสำหรับการสร้างอ็อบเจกต์ โดยกำหนดชุดของคุณลักษณะ (ข้อมูล) และเมธอด (ฟังก์ชัน) ที่อ็อบเจกต์ซึ่งสร้างจากคลาสนั้นสามารถใช้งานได้ ให้นึกว่าคลาสคือแม่แบบที่ระบุว่าอ็อบเจกต์จะมีลักษณะและพฤติกรรมอย่างไร

ไวยากรณ์:

```
class ClassName:
    # Class attributes
    # Class methods
```

ตัวอย่าง:

```
class Car:
    # Class attribute
    wheels = 4

    # Initializer method (constructor)
    def __init__(self, make, model, year):
        self.make = make  # Instance attribute
        self.model = model  # Instance attribute
        self.year = year  # Instance attribute

    # Method
    def start_engine(self):
        return f"The engine of the {self.year} {self.make} {self.model} is
            now running."

    # Method
    def stop_engine(self):
        return f"The engine of the {self.year} {self.make} {self.model} is
            now off."
```

Listing 11.1: Example of a Class

อ็อบเจกต์ (Objects)

อ็อบเจกต์คือตัวอย่าง (instance) ของคลาส เมื่อคุณสร้างอ็อบเจกต์ขึ้นมา นั่นคือคุณกำลังทำการ instantiate คลาส อ็อบเจกต์มีสถานะและพฤติกรรมตามที่กำหนดไว้ในแอตทริบิวต์และเมธอดของคลาส

การสร้างอ็อบเจกต์:

ตัวอย่าง:

```
# Creating an instance of the Car class
my_car = Car("Toyota", "Corolla", 2020)
```

Listing 11.2: Creating an Object

การเข้าถึงแอตทริบิวต์และเมธอด:

```
# Accessing instance attributes
print(my_car.make)  # Output: Toyota
print(my_car.model)  # Output: Corolla
print(my_car.year)  # Output: 2020

# Calling instance methods
print(my_car.start_engine())  # Output: The engine of the 2020 Toyota
    Corolla is now running.
print(my_car.stop_engine())  # Output: The engine of the 2020 Toyota
    Corolla is now off.
```

Listing 11.3: Accessing Attributes and Methods

ข้อดีของการใช้คลาสและอ็อบเจกต์

- **ความเป็นโมดูล (Modularity)**: คลาสช่วยให้สามารถแบ่งปัญหาที่ซับซ้อนออกเป็นส่วนย่อยที่จัดการได้ง่าย
- **การนำกลับมาใช้ (Reusability)**: เมื่อกำหนดคลาสไว้แล้ว สามารถนำไปสร้างอ็อบเจกต์ได้หลายตัว ช่วยลดความซ้ำซ้อน
- **การดูแลรักษา (Maintainability)**: การเปลี่ยนแปลงภายในคลาสจะมีผลกับอ็อบเจกต์ทั้งหมด ทำให้การปรับปรุงระบบง่ายขึ้น
- **การห่อหุ้ม (Encapsulation)**: คลาสช่วยรวมข้อมูลและฟังก์ชันไว้ด้วยกัน พร้อมจำกัดการเข้าถึงบางส่วนเพื่อปกป้องความถูกต้องของข้อมูล

ตัวอย่างการใช้งานจริง

ลองพิจารณาสถานการณ์ง่าย ๆ ในการจัดการระบบห้องสมุด คุณสามารถสร้างคลาส `Book` เพื่อรวมคุณสมบัติและพฤติกรรมของหนังสือไว้ภายใน

ตัวอย่าง:

```python
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_checked_out = False

    def check_out(self):
        if not self.is_checked_out:
            self.is_checked_out = True
            return f"'{self.title}' has been checked out."
        else:
            return f"'{self.title}' is already checked out."

    def return_book(self):
        if self.is_checked_out:
            self.is_checked_out = False
            return f"'{self.title}' has been returned."
        else:
            return f"'{self.title}' was not checked out."

# Creating book objects
book1 = Book("1984", "George Orwell", "1234567890")
book2 = Book("To Kill a Mockingbird", "Harper Lee", "0987654321")

# Checking out and returning books
print(book1.check_out())  # Output: '1984' has been checked out.
print(book1.return_book())  # Output: '1984' has been returned.
print(book2.check_out())  # Output: 'To Kill a Mockingbird' has been
    checked out.
```

Listing 11.4: Example of a Practical Application

การใช้คลาสและอ็อบเจกต์จะช่วยให้คุณสามารถจัดการหน่วยต่าง ๆ ในโปรแกรมได้อย่างมีประสิทธิภาพ ทำให้โค้ดมีโครงสร้างที่ดี นำกลับมาใช้ได้ และดูแลรักษาง่าย

11.3 การสืบทอด

การสืบทอด

การสืบทอด (Inheritance) ช่วยให้คลาสสามารถรับคุณสมบัติและเมธอดจากคลาสอื่นได้ ซึ่งเป็นกลไกในการสร้างคลาสใหม่จากคลาสที่มีอยู่แล้ว คุณสมบัตินี้ช่วยให้สามารถนำโค้ดกลับมาใช้ใหม่ และสร้างโครงสร้างลำดับชั้น (hierarchical structure) ของคลาสอย่างเป็นธรรมชาติ

คลาสที่ถูกสืบทอดเรียกว่า คลาสแม่ หรือ คลาสฐาน (parent หรือ base class) ซึ่งมักจะประกอบด้วยคุณลักษณะและเมธอดทั่วไปที่ใช้ร่วมกันได้ ส่วนคลาสที่ทำการสืบทอดเรียกว่า คลาสลูก หรือ คลาสที่สืบทอดมา (child หรือ derived class) โดยคลาสลูกจะได้รับคุณสมบัติและพฤติกรรมจากคลาสแม่ และยังสามารถเพิ่มคุณสมบัติเฉพาะของตนเองหรือเขียนเมธอดใหม่เพื่อให้เหมาะสมกับการใช้งานเฉพาะทางได้อีกด้วย

โครงสร้างแบบลำดับชั้นนี้ทำให้นักพัฒนาสามารถสร้างระบบที่ซับซ้อนได้ด้วยโครงสร้างที่ชัดเจน โดยมีฟังก์ชันหลักอยู่ในคลาสฐาน และฟีเจอร์เฉพาะอยู่ในคลาสลูก ส่งผลให้โค้ดสะอาด มีระเบียบ และดูแลรักษาง่าย

ไวยากรณ์:

```
class ParentClass:
    # Parent class attributes and methods

class ChildClass(ParentClass):
    # Child class attributes and methods
```

ตัวอย่าง:

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return "Some sound"

class Dog(Animal):
    def speak(self):
        return f"{self.name} says woof!"

class Cat(Animal):
    def speak(self):
        return f"{self.name} says meow!"

# Creating objects of child classes
dog = Dog("Buddy")
cat = Cat("Whiskers")

print(dog.speak())  # Output: Buddy says woof!
print(cat.speak())  # Output: Whiskers says meow!
```

Listing 11.5: Example of Inheritance

11.4 การห่อหุ้ม

การห่อหุ้ม

การห่อหุ้ม (Encapsulation) เป็นแนวคิดพื้นฐานใน OOP ที่รวมข้อมูล (แอตทริบิวต์) และเมธอด (ฟังก์ชัน) ที่ทำงานกับข้อมูลนั้นไว้ในหน่วยเดียวกัน โดยปกติคือภายในคลาส การห่อหุ้มจะจำกัดการเข้าถึงองค์ประกอบภายในบางส่วนของอ็อบเจกต์ ซึ่งช่วยป้องกันการเปลี่ยนแปลงหรือการใช้งานข้อมูลโดยไม่ได้ตั้งใจ และยังช่วยให้โค้ดเป็นโมดูลและดูแลรักษาได้ง่ายขึ้น

หลักการสำคัญของการห่อหุ้ม

- **แอตทริบิวต์และเมธอดแบบส่วนตัว (Private)**: การกำหนดให้แอตทริบิวต์หรือเมธอดเป็นแบบส่วนตัว โดยใช้ขีดล่างสองตัว (__) หน้าชื่อ จะป้องกันไม่ให้สามารถเข้าถึงได้จากภายนอกคลาส

- **เมธอดแบบสาธารณะ (Public)**: เมธอดประเภทนี้ เช่น getter และ setter ใช้ในการเข้าถึงและปรับเปลี่ยนค่าของแอตทริบิวต์ส่วนตัวในลักษณะที่ควบคุมได้

- **การซ่อนข้อมูล (Information Hiding)**: การห่อหุ้มช่วยซ่อนรายละเอียดของการทำงานภายในอ็อบเจกต์และเปิดเผยเฉพาะส่วนที่จำเป็นผ่านเมธอดสาธารณะ ทำให้ใช้งานง่ายขึ้นและลดความผิดพลาด

ตัวอย่างของการห่อหุ้ม

พิจารณาคลาส `Person` ที่รวมข้อมูลส่วนบุคคล เช่น ชื่อและอายุ โดยข้อมูลเหล่านี้ไม่ควรถูกเปลี่ยนแปลงโดยตรงเพื่อรักษาความถูกต้องของข้อมูล

```python
class Person:
    def __init__(self, name, age):
        self.__name = name  # Private attribute
        self.__age = age    # Private attribute

    # Getter method for name
    def get_name(self):
        return self.__name

    # Setter method for name
    def set_name(self, name):
        self.__name = name

    # Getter method for age
    def get_age(self):
        return self.__age

    # Setter method for age with validation
    def set_age(self, age):
        if age > 0:
            self.__age = age
        else:
            print("Age must be positive")

# Creating an object of Person class
person = Person("Alice", 30)

# Accessing private attributes through getter methods
print(person.get_name())  # Output: Alice
print(person.get_age())   # Output: 30

# Modifying private attributes through setter methods
person.set_name("Bob")
person.set_age(35)

print(person.get_name())  # Output: Bob
print(person.get_age())   # Output: 35

# Attempting to set an invalid age
person.set_age(-5)  # Output: Age must be positive
```

Listing 11.6: Example of Encapsulation

ข้อดีของการห่อหุ้ม

- **ควบคุมข้อมูลได้ดีขึ้น**: สามารถควบคุมการเข้าถึงและปรับปรุงค่าด้วยเมธอด getter และ setter ซึ่งเหมาะสำหรับกรณีที่ต้องตรวจสอบค่าก่อนเปลี่ยน
- **เพิ่มความสามารถในการดูแลรักษา**: การห่อหุ้มช่วยแยกส่วนภายในออกจากภายนอก หากมีการเปลี่ยนแปลงโครงสร้างข้อมูลภายใน ก็สามารถเปลี่ยนเฉพาะภายในคลาสโดยไม่กระทบโค้ดภายนอก
- **เพิ่มความปลอดภัย**: การจำกัดการเข้าถึงข้อมูลภายในอ็อบเจกต์ช่วยป้องกันการเปลี่ยนแปลงที่ไม่ตั้งใจหรือไม่ปลอดภัย
- **อินเทอร์เฟซที่เรียบง่าย**: เปิดเผยเฉพาะเมธอดที่จำเป็น ทำให้การใช้งานคลาสง่ายและปลอดภัย

ตัวอย่างการใช้งานจริง

พิจารณาคลาส `BankAccount` ที่รวมรายละเอียดของบัญชีและมีเมธอดสำหรับฝากและถอนเงิน

ตัวอย่าง:

```python
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.__account_number = account_number  # Private attribute
        self.__balance = balance                # Private attribute

    # Getter method for balance
    def get_balance(self):
        return self.__balance

    # Method to deposit money
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            return f"Deposited {amount}. New balance is {self.__balance}."
        else:
            return "Deposit amount must be positive."

    # Method to withdraw money with validation
    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            return f"Withdrew {amount}. New balance is {self.__balance}."
        else:
            return "Insufficient funds or invalid amount."

# Creating an account object
account = BankAccount("1234567890", 1000)

# Accessing balance through getter method
print(account.get_balance())  # Output: 1000

# Depositing money
print(account.deposit(500))   # Output: Deposited 500. New balance is 1500.

# Withdrawing money
print(account.withdraw(200))  # Output: Withdrew 200. New balance is 1300.

# Attempting to withdraw more than the balance
print(account.withdraw(2000)) # Output: Insufficient funds or invalid
    amount.
```

Listing 11.7: Example of a Practical Application

ด้วยการใช้แนวคิดการห่อหุ้ม คุณสามารถจัดการข้อมูลภายในของอ็อบเจกต์ได้อย่างปลอดภัย โดยทุกการกระทำต่อข้อมูลจะต้องผ่านเมธอดที่ควบคุมไว้ ส่งผลให้ข้อมูลมีความสมบูรณ์และปลอดภัยตลอดกระบวนการ

11.5 พอลิมอร์ฟิซึม

พอลิมอร์ฟิซึม

พอลิมอร์ฟิซึม (Polymorphism) เป็นแนวคิดหลักใน OOP ซึ่งช่วยให้เมธอดสามารถทำงานแตกต่างกันตามประเภทของอ็อบเจกต์ที่ถูกเรียกใช้งาน แม้จะใช้ชื่อเมธอดเดียวกันก็ตาม คำว่า ”Polymorphism” แปลว่า ”หลายรูปแบบ”

ซึ่งหมายถึงความสามารถของอ็อบเจกต์ที่แตกต่างกันในการตอบสนองต่อคำสั่งเดียวกันในลักษณะที่ต่างกัน

พอลิมอร์ฟิซึมช่วยเพิ่มความยืดหยุ่นและความสามารถในการขยายของโค้ด โดยเปิดทางให้สามารถใช้ส่วนติดต่อเดียวกัน (interface) กับอ็อบเจกต์หลากหลายชนิดที่มีพฤติกรรมต่างกัน

ประเภทหลักของพอลิมอร์ฟิซึม

- **การเขียนเมธอดซ้ำ (Method Overriding)** หรือพอลิมอร์ฟิซึมขณะรันไทม์ (Runtime Polymorphism): เกิดขึ้นเมื่อคลาสลูกเขียนเมธอดใหม่โดยใช้ชื่อเดียวกับในคลาสแม่ เพื่อกำหนดพฤติกรรมเฉพาะของตนเอง
- **การใช้เมธอดซ้ำชื่อ (Method Overloading)** หรือพอลิมอร์ฟิซึมขณะคอมไพล์ไทม์ (Compile-time Polymorphism): Python ไม่รองรับโดยตรง แต่สามารถเลียนแบบได้โดยกำหนดพารามิเตอร์ที่แตกต่างกันในเมธอด

ตัวอย่างการเขียนเมธอดซ้ำ

พิจารณาสถานการณ์ที่สัตว์แต่ละชนิดส่งเสียงแตกต่างกัน เมธอด `speak` ถูกกำหนดใหม่ในแต่ละคลาสสัตว์

ตัวอย่าง:

```
class Animal:
    def speak(self):
        raise NotImplementedError("Subclass must implement abstract method"
            )

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

# Polymorphism in action
def make_animal_speak(animal):
    print(animal.speak())

dog = Dog()
cat = Cat()

make_animal_speak(dog)  # Output: Woof!
make_animal_speak(cat)  # Output: Meow!
```

Listing 11.8: Example of Method Overriding

ในตัวอย่างนี้ ฟังก์ชัน `make_animal_speak` สามารถรับอ็อบเจกต์ใด ๆ ที่สืบทอดจากคลาส `Animal` และเรียกเมธอด `speak` ได้อย่างถูกต้อง ซึ่งแสดงถึงพอลิมอร์ฟิซึมขณะรันไทม์ โดยคำสั่งเดียวกันให้ผลลัพธ์ต่างกันขึ้นอยู่กับประเภทของอ็อบเจกต์

ตัวอย่างการใช้พอลิมอร์ฟิซึมกับฟังก์ชัน

พอลิมอร์ฟิซึมยังสามารถใช้กับฟังก์ชันที่ทำงานกับอ็อบเจกต์หลากหลายชนิดที่มีเมธอดร่วม เช่น การคำนวณพื้นที่ของรูปร่างที่แตกต่างกัน

ตัวอย่าง:

```python
class Shape:
    def area(self):
        raise NotImplementedError("Subclass must implement abstract method"
            )

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        import math
        return math.pi * (self.radius ** 2)

# List of different shapes
shapes = [Rectangle(10, 20), Circle(5)]

for shape in shapes:
    print(f"Area: {shape.area()}")
    # Output:
    # Area: 200
    # Area: 78.53981633974483
```

Listing 11.9: Example of Polymorphism with Functions

ในตัวอย่างนี้ เมธอด `area` ถูกเรียกใช้กับอ็อบเจกต์ที่มีชนิดต่างกัน (`Rectangle` และ `Circle`) โดยแต่ละคลาสมีการกำหนดเมธอด `area` เป็นของตนเอง แสดงให้เห็นถึงพอลิมอร์ฟิซึมที่แท้จริง

ข้อดีของพอลิมอร์ฟิซึม

- **ยืดหยุ่นและขยายง่าย**: ฟังก์ชันหรือเมธอดเดียวสามารถทำงานร่วมกับอ็อบเจกต์หลายชนิดได้ ทำให้โค้ดยืดหยุ่นและรองรับการขยายได้ง่าย
- **สามารถนำโค้ดกลับมาใช้**: ช่วยให้สามารถใช้โค้ดซ้ำกับอ็อบเจกต์ต่างชนิดได้โดยไม่ต้องเขียนซ้ำ
- **ดูแลรักษาโค้ดได้ง่ายขึ้น**: การเพิ่มอ็อบเจกต์หรือชนิดใหม่สามารถทำได้โดยไม่กระทบกับโค้ดเดิมมากนัก

ตัวอย่างการใช้งานจริง

พิจารณาแอปพลิเคชันที่ต้องจัดการการชำระเงินจากแหล่งต่าง ๆ เช่น บัตรเครดิต, PayPal, และการโอนเงินผ่านธนาคาร โดยกำหนด interface ร่วมสำหรับการประมวลผลการชำระเงิน และให้แต่ละประเภทกำหนดการทำงานเฉพาะของตนเอง

ตัวอย่าง:

```python
class Payment:
    def process_payment(self, amount):
        raise NotImplementedError("Subclass must implement abstract method")

class CreditCardPayment(Payment):
    def process_payment(self, amount):
        return f"Processing credit card payment of {amount}"

class PayPalPayment(Payment):
    def process_payment(self, amount):
        return f"Processing PayPal payment of {amount}"

class BankTransferPayment(Payment):
    def process_payment(self, amount):
        return f"Processing bank transfer payment of {amount}"

# Function to process payments
def process_payment(payment, amount):
    print(payment.process_payment(amount))

# Creating payment objects
credit_card_payment = CreditCardPayment()
paypal_payment = PayPalPayment()
bank_transfer_payment = BankTransferPayment()

# Processing different types of payments
process_payment(credit_card_payment, 100)  # Output: Processing credit card payment of 100
process_payment(paypal_payment, 200)       # Output: Processing PayPal payment of 200
process_payment(bank_transfer_payment, 300) # Output: Processing bank transfer payment of 300
```

Listing 11.10: Example of a Practical Application

จากตัวอย่างนี้ ฟังก์ชัน `process_payment` แสดงให้เห็นถึงพอลิมอร์ฟิซึม โดยสามารถรับอ็อบเจกต์ใด ๆ ที่สืบทอดจากคลาส `Payment` และประมวลผลตามชนิดที่ระบุไว้ได้อย่างเหมาะสม โครงสร้างนี้ช่วยให้สามารถเพิ่มประเภทการชำระเงินใหม่ ๆ ได้โดยไม่ต้องแก้ไขโค้ดที่มีอยู่เดิม

การใช้พอลิมอร์ฟิซึมช่วยให้นักพัฒนาสามารถสร้างโปรแกรมที่ยืดหยุ่น ปรับเปลี่ยนได้ง่าย และรองรับการขยายในอนาคต ซึ่งทำให้โค้ดมีความแข็งแกร่งและดูแลรักษาได้ง่ายยิ่งขึ้น

11.6 ตัวอย่างการใช้งานจริง

ส่วนนี้เน้นตัวอย่างการใช้งานจริงเพื่อแสดงให้เห็นถึงการนำแนวคิดของการเขียนโปรแกรมเชิงวัตถุ (OOP) เช่น การสืบทอด (inheritance), การห่อหุ้ม (encapsulation), และพอลิมอร์ฟิซึม (polymorphism) มาประยุกต์ใช้ในภาษา Python ตัวอย่างเหล่านี้ถูกออกแบบมาเพื่อให้ผู้อ่านเข้าใจได้ชัดเจนว่าแนวคิดเหล่านี้สามารถนำไปใช้ในสถานการณ์จริงได้อย่างไร และแสดงให้เห็นถึงพลังและความยืดหยุ่นของ OOP

11.6.1 ตัวอย่าง: การสร้างคลาสบัญชีธนาคารแบบง่าย

โดยใช้หลักการของการเขียนโปรแกรมเชิงวัตถุ เราสามารถสร้างคลาส `BankAccount` แบบง่ายที่มีเมธอดสำหรับ `(deposit)` และ `(withdraw)` ได้

ตัวอย่าง:

```python
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"Deposited {amount}. New balance is {self.balance}."
        else:
            return "Deposit amount must be positive."

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return f"Withdrew {amount}. New balance is {self.balance}."
        else:
            return "Insufficient funds or invalid amount."

    def get_balance(self):
        return self.balance

# Creating an account
account = BankAccount("1234567890", 1000)

# Depositing money
print(account.deposit(500))  # Output: Deposited 500. New balance is 1500.

# Withdrawing money
print(account.withdraw(200))  # Output: Withdrew 200. New balance is 1300.

# Checking balance
print(account.get_balance())  # Output: 1300
```

Listing 11.11: Example of a Simple Bank Account Class

11.6.2 ตัวอย่าง: การสืบทอดและการเขียนเมธอดซ้ำ

โดยใช้แนวคิดของการเขียนโปรแกรมเชิงวัตถุ เราสามารถสร้างลำดับชั้นของรูปร่างทางเรขาคณิตพร้อมเมธอดสำหรับคำนวณพื้นที่ของแต่ละรูปร่างได้

พิจารณาสถานการณ์ที่คุณต้องการจัดการและคำนวณพื้นที่ของรูปร่างต่าง ๆ โดยใช้พอลิมอร์ฟิซึม คุณสามารถกำหนดส่วนติดต่อร่วมสำหรับคลาส `Shape` และให้แต่ละประเภทของรูปร่างกำหนดการทำงานของตัวเอง

ตัวอย่าง:

```python
class Shape:
    def area(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        import math
        return math.pi * (self.radius ** 2)

# Creating objects
rect = Rectangle(5, 10)
circle = Circle(7)

# Calculating areas
print(f"Rectangle area: {rect.area()}")  # Output: Rectangle area: 50
print(f"Circle area: {circle.area()}")  # Output: Circle area: 153.93804002589985
```

Listing 11.12: Example of a Hierarchy of Geometric Shapes

ในตัวอย่างนี้ คลาส `Shape` ทำหน้าที่เป็นส่วนติดต่อร่วม (interface) ซึ่งกำหนดเมธอด `area` โดยคลาส `Rectangle` และ `Circle` ได้เขียนเมธอดนี้ขึ้นมาใหม่ตามรูปแบบเฉพาะของแต่ละคลาส พอลิมอร์ฟิซึมช่วยให้สามารถคำนวณพื้นที่ของแต่ละรูปร่างโดยใช้เมธอดเดียวกัน แต่ผลลัพธ์ขึ้นอยู่กับประเภทของอ็อบเจกต์

แนวทางนี้ทำให้โค้ดมีความยืดหยุ่นและขยายได้ง่าย คุณสามารถเพิ่มคลาสรูปร่างใหม่ ๆ ได้โดยไม่ต้องแก้ไขโค้ดเดิมมากนัก

11.7 การสาธิตระบบห้องสมุดที่ใช้ OOP

ภาพรวม

นี่คือตัวอย่างขนาดเล็กของโปรเจกต์ระบบห้องสมุดที่ใช้แนวคิดการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming - OOP) โดยส่วนหน้าใช้ FastHTML ส่วนหลังใช้ FastAPI และฐานข้อมูลใช้ SQLite3

การสาธิตนี้ประกอบด้วย 3 ส่วนหลัก:

- **Backend**: ใช้ FastAPI สำหรับจัดการคำขอ API และฐานข้อมูล SQLite3
- **Frontend**: ใช้ HTML template ให้บริการผ่าน FastAPI (ทำงานเหมือน FastHTML)
- **Database**: ใช้ SQLite3 สำหรับจัดเก็บรายการหนังสือ

1. ส่วนหลัง (Backend) ด้วย FastAPI และ SQLite3

เริ่มต้นด้วยการตั้งค่า backend โดยใช้ FastAPI และ SQLite3

โครงสร้างไดเรกทอรี:

```
.
   backend.py
   database.py
   models.py
   templates
       index.html
```

models.py: นิยามคลาส Book

```python
from pydantic import BaseModel

# Pydantic model for the Book
class Book(BaseModel):
    title: str
    author: str
    year: int
    description: str
```

database.py: การตั้งค่าและการทำงานกับฐานข้อมูล

```python
import sqlite3

class Database:
    def __init__(self):
        self.connection = sqlite3.connect('library.db')
        self.cursor = self.connection.cursor()
        self.create_table()

    def create_table(self):
        # Create the books table if it does not exist
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS books
                            (id INTEGER PRIMARY KEY AUTOINCREMENT,
                            title TEXT, author TEXT, year INTEGER,
                               description TEXT)''')
        self.connection.commit()

    def add_book(self, title, author, year, description):
        self.cursor.execute("INSERT INTO books (title, author, year,
           description) VALUES (?, ?, ?, ?)",
                            (title, author, year, description))
        self.connection.commit()

    def get_books(self):
        self.cursor.execute("SELECT * FROM books")
        return self.cursor.fetchall()

    def get_book(self, book_id):
```

```
        self.cursor.execute("SELECT * FROM books WHERE id=?", (book_id,))
        return self.cursor.fetchone()

    def delete_book(self, book_id):
        self.cursor.execute("DELETE FROM books WHERE id=?", (book_id,))
        self.connection.commit()

    def close(self):
        self.connection.close()
```

backend.py: การตั้งค่า FastAPI

```
from fastapi import FastAPI, HTTPException, Request, Form
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
from models import Book
from database import Database

app = FastAPI()
templates = Jinja2Templates(directory="templates")
db = Database()

# Home route to render the book list
@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    books = db.get_books()
    return templates.TemplateResponse("index.html", {"request": request, "
        books": books})

# Add a book (backend endpoint)
@app.post("/add")
async def add_book(title: str = Form(...), author: str = Form(...), year:
    int = Form(...), description: str = Form(...)):
    db.add_book(title, author, year, description)
    return {"message": "Book added successfully"}

# Fetch a single book
@app.get("/books/{book_id}", response_class=HTMLResponse)
async def get_book(book_id: int, request: Request):
    book = db.get_book(book_id)
    if not book:
        raise HTTPException(status_code=404, detail="Book not found")
    return templates.TemplateResponse("book.html", {"request": request, "
        book": book})

# Delete a book
@app.post("/books/{book_id}/delete")
async def delete_book(book_id: int):
    db.delete_book(book_id)
    return {"message": "Book deleted successfully"}

# Close the database connection when the app shuts down
@app.on_event("shutdown")
def shutdown():
    db.close()
```

2. ส่วนหน้า (HTML Templates)

templates/index.html: แสดงรายการหนังสือและเพิ่มหนังสือใหม่

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Library</title>
</head>
<body>
    <h1>Library Books</h1>

    <ul>
        {% for book in books %}
            <li>
                <a href="/books/{{ book[0] }}">{{ book[1] }}</a> - {{ book
                    [2] }} ({{ book[3] }})
                <form method="post" action="/books/{{ book[0] }}/delete"
                    style="display: inline;">
                    <button type="submit">Delete</button>
                </form>
            </li>
        {% endfor %}
    </ul>

    <h2>Add a New Book</h2>
    <form method="post" action="/add">
        <label for="title">Title:</label>
        <input type="text" name="title" required><br><br>
        <label for="author">Author:</label>
        <input type="text" name="author" required><br><br>
        <label for="year">Year:</label>
        <input type="number" name="year" required><br><br>
        <label for="description">Description:</label>
        <textarea name="description" required></textarea><br><br>
        <button type="submit">Add Book</button>
    </form>
</body>
</html>
```

templates/book.html: แสดงรายละเอียดหนังสือหนึ่งเล่ม

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Details</title>
</head>
<body>
    <h1>{{ book[1] }}</h1>
    <p><strong>Author:</strong> {{ book[2] }}</p>
    <p><strong>Year:</strong> {{ book[3] }}</p>
```

```
    <p><strong>Description:</strong> {{ book[4] }}</p>

    <a href="/">Back to Library</a>
</body>
</html>
```

3. การรันโปรเจกต์

ในการรันโปรเจกต์นี้ คุณต้องติดตั้งไลบรารีที่จำเป็นก่อน:

```
pip install fastapi uvicorn jinja2
```

การรันแอป FastAPI ด้วย Uvicorn:

```
uvicorn backend:app --reload
```

จากนั้นให้เปิดเบราว์เซอร์ของคุณและเข้าไปที่ http://127.0.0.1:8000 คุณจะเห็นรายการหนังสือและแบบฟอร์มสำหรับเพิ่มหนังสือใหม่

คำอธิบายระบบ

- **Backend (FastAPI)**:
  - กำหนดเส้นทางสำหรับเพิ่ม ดู และลบหนังสือ
  - ใช้ SQLite3 สำหรับจัดเก็บข้อมูลหนังสือ
  - ใช้ Pydantic สำหรับสร้างโมเดลข้อมูล (Book class)
  - จัดการคำขอ HTTP ด้วย FastAPI
- **Frontend (HTML)**:
  - ใช้ Jinja2 templates แสดงข้อมูลหนังสือที่ได้จาก backend
  - อนุญาตให้ผู้ใช้งานดู เพิ่ม และลบหนังสือผ่านแบบฟอร์มที่ใช้งานง่าย
- **Database (SQLite3)**:
  - จัดเก็บข้อมูลหนังสือในฐานข้อมูล SQLite3
  - จัดการการดำเนินการ CRUD โดยใช้คลาส Database

โปรเจกต์ขนาดเล็กนี้แสดงให้เห็นถึงการสร้างระบบห้องสมุดด้วยแนวคิด OOP โดยใช้ FastAPI, SQLite3 และ HTML templates อย่างมีประสิทธิภาพ

บทที่ 11 โจทย์และแบบฝึกหัด: การเขียนโปรแกรมเชิงวัตถุ (OOP)

11.1 สร้างคลาสพร้อมเมธอดหลายตัว
สร้างคลาส `Car` ที่มีแอตทริบิวต์สำหรับยี่ห้อ รุ่น ปี และเลขไมล์ของรถ พร้อมเมธอดสำหรับอ่านเลขไมล์ เพิ่มเลขไมล์ตามค่าที่ระบุ และรีเซ็ตเลขไมล์ให้เป็นศูนย์

11.2 การใช้งานการสืบทอด (Inheritance)
สร้างคลาสพื้นฐาน `Employee` ที่มีแอตทริบิวต์ `name` และ `salary` แล้วสร้างคลาสลูกชื่อ `Manager` ซึ่งเพิ่มแอตทริบิวต์ `department` และมีเมธอดแสดงรายละเอียดของผู้จัดการรวมถึงแผนก

11.3 การใช้การห่อหุ้ม (Encapsulation)
สร้างคลาส `BankAccount` โดยมีแอตทริบิวต์ส่วนตัวสำหรับ และ พร้อมเมธอดสำหรับฝาก ถอน และตรวจสอบยอดเงิน โดยห้ามเข้าถึงแอตทริบิวต์โดยตรงจากภายนอกคลาส

11.4 พอลิมอร์ฟิซึมกับรูปร่างต่าง ๆ
สร้างคลาสพื้นฐาน `Shape` ที่มีเมธอดนามธรรมชื่อ `area()` แล้วสร้างคลาสลูก `Triangle`, `Square`, และ `Circle` ซึ่งแต่ละคลาสต้องมีแอตทริบิวต์สำหรับคำนวณพื้นที่และกำหนดการทำงานของเมธอด `area()`

11.5 การเขียนเมธอดซ้ำ (Method Overriding)
สร้างคลาสพื้นฐานชื่อ `Instrument` ที่มีเมธอด `play_sound()` จากนั้นสร้างคลาสลูก `Piano` และ `Guitar` ซึ่งเขียนเมธอด `play_sound()` ขึ้นใหม่เพื่อแสดงข้อความที่เฉพาะเจาะจง

11.6 การห่อหุ้มโครงสร้างข้อมูลที่ซับซ้อน
สร้างคลาส `Library` ที่ห่อหุ้มการจัดเก็บหนังสือ โดยแต่ละเล่มใช้ dictionary ที่มี key เป็น `title`, `author`, และ `year` พร้อมเมธอดสำหรับเพิ่ม ลบ และแสดงรายการหนังสือทั้งหมด

11.7 การใช้เมธอดจากคลาสแม่
สร้างคลาส `Person` ที่มีแอตทริบิวต์ `name` และ `age` พร้อมเมธอด `display()` แล้วสร้างคลาสลูก `Student` ที่เพิ่มแอตทริบิวต์ `student_id` และเขียนเมธอด `display()` ใหม่ให้แสดงข้อมูลของนักเรียนทั้งหมด

11.8 การสร้างคลาสนามธรรม (Abstract Base Class)
สร้างคลาสนามธรรมชื่อ `Vehicle` ที่มีเมธอดนามธรรม `fuel_efficiency()` จากนั้นสร้างคลาสลูก `Car` และ `Bike` ที่กำหนดเมธอดนี้ขึ้นเองในแต่ละคลาส

11.9 การใช้ property เพื่อห่อหุ้มข้อมูล
สร้างคลาส `Employee` โดยมีแอตทริบิวต์ส่วนตัวชื่อ `salary` และใช้ `property` สำหรับ getter และ setter โดยกำหนดให้ไม่สามารถกำหนดค่าติดลบได้

11.10 การใช้งาน Multiple Inheritance
สร้างคลาส `FlyingVehicle` ที่มีเมธอด `fly()` และคลาส `Vehicle` ที่มีเมธอด `drive()` แล้วสร้างคลาส `FlyingCar` ที่สืบทอดจากทั้งสองคลาสและสามารถใช้งานได้ทั้ง `fly()` และ `drive()`

ภาคผนวก

ภาคผนวกนี้จัดเตรียมแหล่งข้อมูลเพิ่มเติม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างที่ใช้งานได้จริง เพื่อเสริมความเข้าใจของคุณเกี่ยวกับแนวคิดที่ครอบคลุมในบทที่ 11

A11.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Learning Python* by Mark Lutz
- *Python 3 Object-Oriented Programming* by Dusty Phillips

บทเรียนออนไลน์:

- Python OOP - W3Schools
- Object-Oriented Programming in Python - Real Python

คอร์สออนไลน์:

- Coursera: Object-Oriented Programming in Python by University of Michigan
- Udemy: Python OOP - Object-Oriented Programming for Beginners

A11.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). Python Documentation - Classes. Retrieved from https://docs.python.org/3/tutorial/classes.html
- Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook* (3rd ed.). O'Reilly Media.

A11.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: การสร้างคลาสพื้นฐาน

- Write a Python class named `Person` with attributes `name` and `age`. Include a method to display the person's details.
- Create an instance of the `Person` class and display its attributes.

แบบฝึกหัดที่ 2: การสืบทอด (Inheritance)

- Develop a class `Employee` that inherits from the `Person` class and adds an attribute `employee_id`. Include a method to display the employee's details.
- Create an instance of the `Employee` class and display its attributes and methods.

แบบฝึกหัดที่ 3: การห่อหุ้ม (Encapsulation)

- Write a Python class `BankAccount` with private attributes `account_number` and `balance`. Include methods to deposit, withdraw, and check the balance.

- Create an instance of the `BankAccount` class and test the methods.

แบบฝึกหัดที่ 4: พอลิมอร์ฟิซึม (Polymorphism)

- Create a base class `Shape` with a method `area()`. Develop two subclasses `Rectangle` and `Circle` that override the `area()` method.
- Create instances of `Rectangle` and `Circle` and demonstrate polymorphism by calling the `area()` method on both instances.

A11.4 ตัวอย่างที่ใช้งานได้จริง

Example 1: Basic Class Definition

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}")

# Creating an instance of the Person class
person1 = Person("Alice", 30)
person1.display()
```

Listing 11.13: Basic class definition

Example 2: Inheritance

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}")

class Employee(Person):
    def __init__(self, name, age, employee_id):
        super().__init__(name, age)
        self.employee_id = employee_id

    def display(self):
        super().display()
        print(f"Employee ID: {self.employee_id}")

# Creating an instance of the Employee class
employee1 = Employee("Bob", 28, "E12345")
employee1.display()
```

Listing 11.14: Inheritance example

Example 3: Encapsulation

```python
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.__account_number = account_number
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient balance")

    def get_balance(self):
        return self.__balance

# Creating an instance of the BankAccount class
account = BankAccount("123456789")
account.deposit(1000)
account.withdraw(500)
print("Balance:", account.get_balance())
```

Listing 11.15: Encapsulation example

Example 4: Polymorphism

```python
class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

# Creating instances of Rectangle and Circle
rectangle = Rectangle(4, 5)
circle = Circle(3)

# Demonstrating polymorphism
print("Area of rectangle:", rectangle.area())
print("Area of circle:", circle.area())
```

Listing 11.16: Polymorphism example

บทที่ 12

การเขียนโปรแกรมเชิงฟังก์ชัน

บทนี้นำเสนอภาพรวมอย่างครอบคลุมเกี่ยวกับการเขียนโปรแกรมเชิงฟังก์ชันในภาษา Python โดยครอบคลุมหัวข้อต่าง ๆ และเทคนิคพื้นฐานของแนวทางนี้ เริ่มต้นด้วยการแนะนำแนวคิดการเขียนโปรแกรมเชิงฟังก์ชันที่เน้นการประเมินฟังก์ชันทางคณิตศาสตร์และหลีกเลี่ยงการเปลี่ยนแปลงสถานะหรือผลข้างเคียงต่าง ๆ แนวคิดหลักเช่น ฟังก์ชันบริสุทธิ์ ฟังก์ชันลำดับสูง และความไม่เปลี่ยนแปลงของข้อมูลจะถูกอธิบายอย่างละเอียด พร้อมตัวอย่างเพื่อแสดงหลักการและการประยุกต์ใช้งาน

บทนี้ยังกล่าวถึงการใช้ฟังก์ชันแลมบ์ดา (lambda) โดยแสดงให้เห็นถึงประโยชน์ในการสร้างฟังก์ชันแบบไม่ระบุชื่อเพื่อใช้งานชั่วคราว อธิบายรูปแบบและการทำงานของฟังก์ชัน `map()`, `filter()`, และ `reduce()` พร้อมตัวอย่างที่แสดงการประมวลผลและแปลงข้อมูลอย่างมีประสิทธิภาพ โดยมีตัวอย่างจริงที่รวมการใช้ฟังก์ชันเหล่านี้เพื่อทำงานที่ซับซ้อน ช่วยให้ผู้อ่านเข้าใจและฝึกปฏิบัติได้อย่างชัดเจน

นอกจากนี้ยังมีการกล่าวถึงแนวคิดของฟังก์ชันลำดับสูงที่สามารถรับฟังก์ชันอื่นเป็นพารามิเตอร์หรือส่งคืนฟังก์ชันเป็นผลลัพธ์ รวมถึงการเวียนกลับ (recursion) ซึ่งเป็นเทคนิคอันทรงพลังในการแก้ปัญหาโดยแบ่งเป็นปัญหาย่อย ๆ เช่น การหาค่าแฟกทอเรียลและการสร้างลำดับฟีโบนักชี

โดยรวมแล้วบทนี้ช่วยให้ผู้อ่านเข้าใจการเขียนโปรแกรมเชิงฟังก์ชันในภาษา Python ได้อย่างมั่นคง พร้อมตัวอย่างประกอบที่แสดงการใช้งานแนวคิดและเทคนิคเหล่านี้ในทางปฏิบัติ

12.1 บทนำสู่การเขียนโปรแกรมเชิงฟังก์ชัน

การเขียนโปรแกรมเชิงฟังก์ชัน (Functional Programming) [4, 5] เป็นแนวทางการเขียนโปรแกรมที่มีพื้นฐานมาจากแนวคิดทางคณิตศาสตร์ โดยเฉพาะฟังก์ชันทางคณิตศาสตร์ที่รับอินพุตแล้วคืนค่าผลลัพธ์โดยไม่มีผลข้างเคียง (side effects) จุดเด่นของแนวทางนี้คือการหลีกเลี่ยงการเปลี่ยนแปลงสถานะของตัวแปรหรือข้อมูล ซึ่งหมายความว่า เมื่อมีการกำหนดค่าหนึ่งให้กับตัวแปรแล้ว ค่านั้นจะไม่ถูกเปลี่ยนแปลงภายหลัง (immutability) การใช้ฟังก์ชันที่ไม่มีผลข้างเคียงทำให้โปรแกรมง่ายต่อการทดสอบ วิเคราะห์ และเข้าใจ

โปรแกรมเมอร์ที่ใช้แนวทางเชิงฟังก์ชันจะนิยมใช้ “ฟังก์ชันบริสุทธิ์” (Pure Functions) ซึ่งเป็นฟังก์ชันที่ให้ผลลัพธ์เดียวกันเสมอเมื่อได้รับอินพุตเดียวกัน และไม่มีการเปลี่ยนแปลงข้อมูลภายนอก นอกจากนี้ยังมีการใช้ “ฟังก์ชันลำดับสูง” (Higher-Order Functions) ซึ่งเป็นฟังก์ชันที่สามารถรับฟังก์ชันอื่นเป็นอาร์กิวเมนต์ หรือคืนค่าฟังก์ชันใหม่ได้ ทำให้สามารถเขียนโค้ดที่มีความยืดหยุ่นและนำกลับมาใช้ใหม่ได้ง่าย

การเขียนโปรแกรมเชิงฟังก์ชันได้รับความนิยมในระบบที่ต้องการความเชื่อถือสูง เช่น ระบบประมวลผลข้อมูลขนาดใหญ่ ปัญญาประดิษฐ์ และแอปพลิเคชันแบบกระจาย เพราะช่วยลดข้อผิดพลาดจากสถานะที่เปลี่ยนแปลงได้ อีกทั้งยังเหมาะสำหรับการประมวลผลแบบขนาน (parallel processing) เนื่องจากไม่ต้องจัดการกับการเปลี่ยนแปลงของข้อมูลร่วมกันระหว่างหลายเธรดหรือโปรเซส

12.2 แนวคิดหลักของการเขียนโปรแกรมเชิงฟังก์ชัน

แนวคิดหลักของการเขียนโปรแกรมเชิงฟังก์ชัน:

- **ฟังก์ชันบริสุทธิ์ (Pure functions)** เป็นหัวใจหลักของการเขียนโปรแกรมเชิงฟังก์ชัน โดยจะให้ผลลัพธ์เดียวกันเสมอเมื่อได้รับอินพุตเดียวกัน และไม่มีผลข้างเคียง ทำให้โปรแกรมเข้าใจง่ายและดีบักได้ง่ายขึ้น
- **ฟังก์ชันลำดับสูง (Higher-order functions)** เพิ่มพลังให้กับโปรแกรมเชิงฟังก์ชัน โดยสามารถรับฟังก์ชันอื่นเป็นอาร์กิวเมนต์หรือส่งคืนฟังก์ชันเป็นผลลัพธ์ได้ ช่วยให้เขียนโค้ดได้ยืดหยุ่นและเป็นนามธรรมมากขึ้น
- **ความไม่เปลี่ยนแปลง (Immutability)** หมายถึงโครงสร้างข้อมูลจะไม่สามารถเปลี่ยนแปลงได้หลังจากถูกสร้างขึ้น แทนที่จะเปลี่ยนข้อมูลเดิม จะสร้างข้อมูลเวอร์ชันใหม่ ทำให้สถานะของโปรแกรมชัดเจนและสม่ำเสมอ
- **ฟังก์ชันเป็นพลเมืองชั้นหนึ่ง (First-class functions)** หมายความว่าในภาษา Python ฟังก์ชันสามารถถูกกำหนดให้กับตัวแปร ส่งเป็นอาร์กิวเมนต์ และส่งกลับจากฟังก์ชันอื่นได้ ช่วยเพิ่มความยืดหยุ่นและการนำกลับมาใช้ซ้ำ

12.2.1 ฟังก์ชันบริสุทธิ์(Pure functions)

ฟังก์ชันบริสุทธิ์คือฟังก์ชันที่เมื่อได้รับอินพุตเดียวกัน จะให้ผลลัพธ์เหมือนเดิมเสมอ และไม่มีผลข้างเคียงใด ๆ (เช่น ไม่เปลี่ยนแปลงสถานะหรือข้อมูลภายนอกฟังก์ชัน)

```python
def add(a, b):
    return a + b

print(add(2, 3))  # Output: 5
print(add(2, 3))  # Output: 5 (same input, same output)
```

Listing 12.1: Example of a Pure Function

12.2.2 ฟังก์ชันลำดับสูง(Higher-order functions)

ฟังก์ชันลำดับสูงคือฟังก์ชันที่สามารถรับฟังก์ชันอื่นเป็นอาร์กิวเมนต์ หรือส่งฟังก์ชันเป็นผลลัพธ์ได้

```python
def apply_function(func, value):
    return func(value)

def square(x):
    return x * x

print(apply_function(square, 5))  # Output: 25
```

Listing 12.2: Example of a Higher-Order Function

12.2.3 ความไม่เปลี่ยนแปลงของข้อมูล(Immutability)

ความไม่เปลี่ยนแปลงหมายถึงข้อมูลไม่สามารถถูกเปลี่ยนแปลงได้หลังจากถูกสร้างขึ้น โดยจะสร้างข้อมูลใหม่แทนที่จะเปลี่ยนแปลงข้อมูลเดิม

```
# Immutable example with strings
name = "Alice"
new_name = name.upper()

print(name)      # Output: Alice (original string is unchanged)
print(new_name)  # Output: ALICE
```

Listing 12.3: Example of Immutability

12.2.4 ฟังก์ชันเป็นพลเมืองชั้นหนึ่ง(First-class functions)

ในภาษา Python ฟังก์ชันถือเป็นพลเมืองชั้นหนึ่ง หมายความว่าสามารถส่งเป็นอาร์กิวเมนต์ ส่งกลับจากฟังก์ชัน และกำหนดให้กับตัวแปรได้

```
def greet(name):
    return f"Hello, {name}!"

say_hello = greet
print(say_hello("Alice"))  # Output: Hello, Alice!
```

Listing 12.4: Example of First-Class Functions

12.3 ฟังก์ชันแลมบ์ดา (Lambda Functions)

ฟังก์ชันแลมบ์ดาเป็นฟังก์ชันแบบไม่ระบุชื่อ (anonymous function) ที่สร้างขึ้นโดยใช้คีย์เวิร์ด `lambda` ซึ่งเหมาะสำหรับฟังก์ชันที่สั้นและเรียบง่ายที่ใช้ชั่วคราว

```
lambda arguments: expression
```

Listing 12.5: Syntax for Lambda Functions

```
# Lambda function for adding two numbers
add = lambda x, y: x + y
print(add(3, 5))  # Output: 8

# Using lambda with map
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x * x, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

Listing 12.6: Example of Lambda Functions

12.4 Map

ฟังก์ชัน `map()`

ฟังก์ชัน `map()` ใช้ในการนำฟังก์ชันที่กำหนดไปประยุกต์ใช้กับแต่ละสมาชิกใน iterable (เช่น รายการ list) และคืนค่าตัววนซ้ำ (iterator)

```
map(function, iterable)
```

Listing 12.7: Syntax for `map()` Function

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

Listing 12.8: Example of `map()` Function

12.5 Filter

ฟังก์ชัน `filter()` สร้างตัววนซ้ำ (iterator) จากสมาชิกของ iterable ที่ฟังก์ชันให้ค่าผลลัพธ์เป็น `True`

```
filter(function, iterable)
```

Listing 12.9: Syntax for `filter()` Function

```
numbers = [1, 2, 3, 4, 5]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4]
```

Listing 12.10: Example of `filter()` Function

12.6 Reduce

ฟังก์ชัน `reduce()` จากมอดูล `functools` ใช้สำหรับประยุกต์ฟังก์ชันที่มีอาร์กิวเมนต์สองตัวกับสมาชิกของ iterable แบบสะสมจากซ้ายไปขวา เพื่อให้ได้ค่าผลลัพธ์สุดท้ายเพียงค่าเดียว

```
from functools import reduce
reduce(function, iterable)
```

Listing 12.11: Syntax for `reduce()` Function

```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]
sum_of_numbers = reduce(lambda x, y: x + y, numbers)
print(sum_of_numbers)  # Output: 15
```

Listing 12.12: Example of `reduce()` Function

ตัวอย่าง: การใช้ Map, Filter และ Reduce ร่วมกัน

การรวมฟังก์ชัน `map()`, `filter()` และ `reduce()` เพื่อประมวลผลข้อมูล

```python
from functools import reduce

# List of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using map to square each number
squared_numbers = map(lambda x: x ** 2, numbers)

# Using filter to keep only even numbers
even_squared_numbers = filter(lambda x: x % 2 == 0, squared_numbers)

# Using reduce to sum the even squared numbers
sum_of_even_squared_numbers = reduce(lambda x, y: x + y,
    even_squared_numbers)
print(sum_of_even_squared_numbers)  # Output: 220
```

Listing 12.13: Combining map filter and reduce to Process Data

ตัวอย่าง: การสร้างฟังก์ชันลำดับสูงด้วยตนเอง

การสร้างฟังก์ชันลำดับสูงที่นำฟังก์ชันที่กำหนดมาใช้สองครั้ง

```python
def apply_twice(func, value):
    return func(func(value))

def increment(x):
    return x + 1

print(apply_twice(increment, 5))  # Output: 7
```

Listing 12.14: Example of a Higher-Order Function Applying a Given Function Twice

ตัวอย่าง: การใช้ฟังก์ชันเป็นค่าที่ส่งกลับ

การสร้างฟังก์ชันที่ส่งกลับฟังก์ชันอื่น

```python
def create_multiplier(n):
    return lambda x: x * n

double = create_multiplier(2)
triple = create_multiplier(3)

print(double(5))  # Output: 10
print(triple(5))  # Output: 15
```

Listing 12.15: Example of a Function that Returns Another Function

12.7 การเวียนกลับ (Recursion)

การเวียนกลับ (Recursion) เป็นเทคนิคที่ฟังก์ชันเรียกใช้งานตัวเองเพื่อแก้ปัญหา เป็นเครื่องมือที่ทรงพลังสำหรับการแบ่งปัญหาออกเป็นปัญหาย่อยที่คล้ายกันและง่ายต่อการจัดการ

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

Listing 12.16: Example of Recursion

ตัวอย่างเชิงปฏิบัติ: ลำดับฟีโบนักชีแบบเวียนกลับ

การสร้างฟังก์ชันเวียนกลับเพื่อสร้างลำดับฟีโบนักชี

```python
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

for i in range(10):
    print(fibonacci(i), end=" ")  # Output: 0 1 1 2 3 5 8 13 21 34
```

Listing 12.17: Example of a Recursive Function to Generate Fibonacci Numbers

บทที่ 12 โจทย์และแบบฝึกหัด: การเขียนโปรแกรมเชิงฟังก์ชัน (Functional Programming)

12.1 ฟังก์ชันบริสุทธิ์และผลข้างเคียง
สร้างฟังก์ชันบริสุทธิ์ชื่อ `calculate_total_price` ที่รับลิสต์ราคาสินค้าและอัตราภาษี แล้วคืนค่าราคารวมหลังรวมภาษี โดยไม่เปลี่ยนแปลงลิสต์ต้นฉบับและไม่มีผลข้างเคียง

12.2 ฟังก์ชันลำดับสูงสำหรับการกรองข้อมูล
เขียนฟังก์ชันชื่อ `filter_list` ที่รับลิสต์และฟังก์ชันเป็นอาร์กิวเมนต์ แล้วส่งคืนลิสต์ใหม่ที่มีเฉพาะค่าที่ผ่านเงื่อนไขของฟังก์ชัน

12.3 การใช้ทูเพิลกับความไม่เปลี่ยนแปลง
กำหนดทูเพิลของจำนวนเต็ม สร้างฟังก์ชัน `multiply_elements` ที่คูณสมาชิกแต่ละตัวด้วยค่าคงที่ และคืนค่าทูเพิลใหม่โดยไม่เปลี่ยนแปลงต้นฉบับ

12.4 การใช้ฟังก์ชันเป็นพลเมืองชั้นหนึ่งในการแปลงข้อมูล
สร้างฟังก์ชันชื่อ `apply_transformation` ที่รับฟังก์ชันและลิสต์ของสตริง แล้วประยุกต์ฟังก์ชันนั้นกับแต่ละสตริง และคืนค่าลิสต์ที่แปลงแล้ว

12.5 การใช้ฟังก์ชันแลมบ์ดาเพื่อจัดเรียงข้อมูล
ใช้ฟังก์ชันแลมบ์ดาในการจัดเรียงลิสต์ของพจนานุกรม โดยอิงจากค่าของคีย์ที่ระบุ

12.6 การใช้ `map` แปลงหน่วยอุณหภูมิ
สร้างฟังก์ชันที่ใช้ `map` แปลงค่าจากองศาเซลเซียสเป็นฟาเรนไฮต์ในลิสต์อุณหภูมิ

12.7 การใช้ `filter` เพื่อหาจำนวนเฉพาะ
เขียนฟังก์ชันที่ใช้ `filter` เพื่อหาจำนวนเฉพาะทั้งหมดจากลิสต์ของตัวเลข

12.8 การใช้ `reduce` เพื่อคำนวณผลคูณ
ใช้ `reduce` เพื่อคำนวณผลคูณของตัวเลขทั้งหมดในลิสต์

12.9 การรวม `map`, `filter`, และ `reduce` เพื่อหาผลรวม
ใช้ `map` สร้างกำลังสองของตัวเลข ใช้ `filter` เลือกเฉพาะจำนวนคู่ แล้วใช้ `reduce` หาผลรวมของค่าที่ได้

12.10 เวียนกลับเพื่อหาผลรวมของลิสต์
เขียนฟังก์ชันแบบเวียนกลับเพื่อหาผลรวมของสมาชิกในลิสต์

12.11 เวียนกลับเพื่อหาค่า GCD
เขียนฟังก์ชันเวียนกลับเพื่อคำนวณหาตัวหารร่วมมาก (GCD) ของจำนวนสองจำนวน

12.12 เวียนกลับเพื่อแก้ปัญหา Tower of Hanoi
เขียนฟังก์ชันแบบเวียนกลับเพื่อแก้ปัญหา Tower of Hanoi สำหรับจำนวนแผ่นดิสก์ `n` แผ่น

ภาคผนวก

ภาคผนวกนี้ให้ทรัพยากรเสริม แหล่งอ้างอิง แบบฝึกหัด และตัวอย่างการใช้งานจริง เพื่อเสริมสร้างความเข้าใจในแนวคิดที่ครอบคลุมในบทที่ 12

A12.1 แหล่งข้อมูลเพิ่มเติม

หนังสือ:

- *Functional Programming in Python* โดย David Mertz
- *Python Cookbook* โดย David Beazley และ Brian K. Jones

บทเรียนออนไลน์:

- Functional Programming in Python - Real Python
- Python Functional Programming - GeeksforGeeks

คอร์สเรียน:

- Coursera: Functional Programming Principles in Scala โดย École Polytechnique Fédérale de Lausanne
- edX: Introduction to Functional Programming โดย Delft University of Technology

A12.2 แหล่งอ้างอิง

- Python Software Foundation. (2024). Python Documentation - Functional Programming. สืบค้นจาก https://docs.python.org/3/howto/functional.html
- Mertz, D. (2015). *Functional Programming in Python*. O'Reilly Media.

A12.3 แบบฝึกหัด

แบบฝึกหัดที่ 1: ฟังก์ชันแลมบ์ดา

- เขียนฟังก์ชันแลมบ์ดาที่เพิ่มค่า 10 ให้กับตัวเลขที่กำหนด
- สร้างลิสต์ของตัวเลข และใช้ฟังก์ชันแลมบ์ดาเพื่อกรองเฉพาะเลขคู่

แบบฝึกหัดที่ 2: Map, Filter และ Reduce

- ใช้ `map` กับฟังก์ชันแลมบ์ดาเพื่อยกกำลังสองของตัวเลขแต่ละตัวในลิสต์
- ใช้ `filter` เพื่อคัดกรองค่าที่มากกว่า 5 จากลิสต์
- ใช้ `reduce` จากมอดูล `functools` เพื่อคำนวณผลคูณของตัวเลขทั้งหมดในลิสต์

แบบฝึกหัดที่ 3: List Comprehensions

- เขียน list comprehension เพื่อสร้างลิสต์ของเลขยกกำลังสองจาก 1 ถึง 10

- เขียน list comprehension เพื่อกรองตัวอักษรที่ไม่ใช่สระออกจากสตริง

แบบฝึกหัดที่ 4: ฟังก์ชันลำดับสูง (Higher-Order Functions)

- เขียนฟังก์ชัน `apply_twice` ที่รับฟังก์ชันและอาร์กิวเมนต์ แล้วใช้ฟังก์ชันนั้นสองครั้ง

- พัฒนาฟังก์ชัน `compose` ที่รับฟังก์ชัน `f` และ `g` แล้วส่งกลับฟังก์ชันใหม่ที่ใช้ `f` กับผลลัพธ์ของ `g`

A12.4 ตัวอย่างการใช้งานจริง

ตัวอย่างที่ 1: ฟังก์ชันแลมบ์ดา

```
# Lambda function to add 10 to a given number
add_ten = lambda x: x + 10
print(add_ten(5))  # Output: 15

# Filtering even numbers using lambda function
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6, 8, 10]
```

Listing 12.18: Lambda function examples

ตัวอย่างที่ 2: การใช้ Map, Filter และ Reduce

```
from functools import reduce

# Using map to square each number in a list
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]

# Using filter to extract numbers greater than 5
numbers = [1, 2, 3, 6, 7, 8, 10]
filtered_numbers = list(filter(lambda x: x > 5, numbers))
print(filtered_numbers)  # Output: [6, 7, 8, 10]

# Using reduce to find the product of all numbers in a list
numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 120
```

Listing 12.19: Using Map Filter and Reduce

ตัวอย่างที่ 3: List Comprehensions

```
# List comprehension to create a list of squares
squares = [x ** 2 for x in range(1, 11)]
print(squares)  # Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

# List comprehension to filter out vowels from a string
string = "Hello, World!"
vowels = "aeiouAEIOU"
filtered_string = ''.join([char for char in string if char not in vowels])
print(filtered_string)  # Output: "Hll, Wrld!"
```

Listing 12.20: List comprehension examples

ตัวอย่างที่ 4: ฟังก์ชันลำดับสูง

```
# Function to apply another function twice
def apply_twice(func, arg):
    return func(func(arg))

# Example usage
print(apply_twice(lambda x: x + 2, 10))  # Output: 14

# Function to compose two functions
def compose(f, g):
    return lambda x: f(g(x))

# Example usage
def add_three(x):
    return x + 3

def multiply_by_two(x):
    return x * 2

composed_function = compose(add_three, multiply_by_two)
print(composed_function(5))  # Output: 13 (5 * 2 + 3)
```

Listing 12.21: Higher-order function examples

บรรณานุกรม

[1] Thomas H. Cormen et al. *Introduction to Algorithms*. Illustrated reprint. Accessed: 2025-06-19. MIT Press, 2001, p. 1180. ISBN: 0262032937. URL: https://books.google.com/books/about/Introduction_To_Algorithms.html?id=NLngYyWFl_YC.

[2] Guido van Rossum and Python Development Team. *Python Tutorial 3.11.3*. Paperback, 166 pp. Lulu.com, May 12, 2023. 166 pp. ISBN: 9781312571655. DOI: ISBN=978-1312571655. URL: https://www.google.co.th/books/edition/Python_Tutorial_3_11_3/J93uzwEACAAJ.

[3] Refsnes Data AS. *Python Tutorial*. Accessed: 19 June 2025. W3Schools. 2025. URL: https://www.w3schools.com/python/.

[4] Python Software Foundation. *Functional Programming HOWTO*. Accessed: 2025-06-19. 2024. URL: https://docs.python.org/3/howto/functional.html.

[5] GeeksforGeeks. *Functional Programming in Python*. https://www.geeksforgeeks.org/python/functional-programming-in-python/. Last updated 02-Jan-2025; accessed 19-Jun-2025. 2025.