

Programming expert with Python

เชี่ยวชาญการเขียนโปรแกรมด้วยไพธอน



ผศ.ดร. สุชาติ คุ้มมะณี

คำนำ

ภาษาไพธอน (Python Language) เป็นภาษาที่ถูกออกแบบและพัฒนามาเพื่อให้ผู้เรียนสามารถเรียนรู้ได้ง่าย รวดเร็ว กระทัดรัด และมีประสิทธิภาพสูง โดยการนำเอาคุณลักษณะเด่นๆ ของภาษาอื่นๆ มาเป็นพื้นฐานในการพัฒนาต่อยอด เช่น ภาษา C, C++, Java, Perl, ABC, Modula-3, Icon, Matlab, ANSI C, Lisp, Smalltalk และ Tcl เป็นต้น ไพธอนจึงถูกเรียกว่าเป็นภาษาที่มีหลายระบบหัวหน้า หรือ หลายมุมมอง (multi-paradigm languages) ซึ่งเกิดจากการผสมผสานรวมเอาแนวความคิดในการพัฒนาซอฟต์แวร์แบบต่างๆ เช่น ไวด์ไวยกันให้อยู่ในตัวของไพธอน คือ การเขียนโปรแกรมเชิงวัตถุ (Object-oriented programming), การเขียนโปรแกรมเชิงโครงสร้าง (Structured programming), การโปรแกรมเชิงฟังชัน (Functional programming) และการเขียนโปรแกรมเชิงลักษณะ (Aspect-oriented programming)

ภาษาไพธอนเป็นภาษาที่ได้รับการพัฒนาต่อยอดจากนักพัฒนาโปรแกรมทั่วโลก ส่งผลให้ภาษาดังกล่าวมีความสามารถสูงและรองรับงานด้านต่างๆ ได้มากมายอาทิเช่น งานด้านวิทยาศาสตร์ วิศวกรรมศาสตร์ ระบบฐานข้อมูล เกมส์และแอปพลิเคชัน เครือข่ายคอมพิวเตอร์ เว็บแอปพลิเคชัน และนิยมใช้สำหรับประกอบการเรียนการสอนในต่างประเทศตั้งแต่ระดับมัธยมถึงมหาวิทยาลัย เห็นได้จากมีหน่วยงานสำคัญๆ ของโลกนำเอาภาษาไพธอนไปพัฒนางานของตนมากมาย เช่น นาซ่า (NASA), กูเกิล (Google), หน่วยงานของสหรัฐ (USA Central Intelligence Agency :CIA), IBM และอื่นๆ สำหรับประเทศไทยกำลังเพิ่มมากขึ้นเรื่อยๆ เช่น มหาวิทยาลัยมหิดล เกษตรศาสตร์ จุฬาลงกรณ์ และพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นต้น หนังสือเล่มนี้จึงเหมาะสมอย่างยิ่งสำหรับเป็นพื้นฐานสำหรับผู้ที่ต้องการเริ่มต้นเรียนรู้ในการเขียนโปรแกรมไปจนถึงผู้เชี่ยวชาญที่ต้องการนำไปประยุกต์ใช้เข้ากับงานของตนเอง หนังสือเล่มนี้ไม่ใช่หนังสือไพธอนที่ดีที่สุด ดังนั้นถ้ามีข้อผิดพลาดประการใดเกิดขึ้น ผู้เขียนขออภัยรับไว้ทั้งหมด

ผู้เขียนขอสงวนลิขสิทธิ์ในหนังสือเล่มนี้เพื่อใช้เป็นวิทยาทานเท่านั้น ห้ามผู้ใด จำหน่าย พิมพ์ เพื่อขาย ให้ดาวน์โหลดโดยคิดค่าบริการ หรือใช้ในเชิงพาณิชย์ทั้งสิ้น แต่อนุญาตให้แจกจ่ายได้

ท้ายนี้ผู้เขียนหวังเป็นอย่างยิ่งว่าผู้อ่านจะได้รับประโยชน์จากหนังสือเล่มนี้ตามสมควร หากมีข้อแนะนำกรุณาแจ้งให้ผู้เขียนได้ทราบ เพื่อจะได้นำไปปรับปรุงแก้ไขต่อไป จึงขอบคุณมา ณ โอกาสนี้

ผศ.ดร. สุชาติ คุ้มมะณี

Email: suchart.k@msu.ac.th

แนะนำเกี่ยวกับหนังสือ

จุดเด่นของหนังสือเล่มนี้ ?

หนังสือ “เชี่ยวชาญการเขียนโปรแกรมด้วยภาษาไพธอน (Programming expert with Python)” ที่ทำนักเรียนเรียนขึ้นมาเพื่อตอบสนองกับผู้ที่ต้องการเขียนโปรแกรมสำหรับแก้ปัญหาต่างๆ ด้วยระบบคอมพิวเตอร์ โดยหนังสือเล่มดังกล่าวนี้ เน้นการพัฒนาโปรแกรมด้วยภาษาไพธอน ซึ่งเป็นที่ยอมรับกันทั่วโลกว่าเป็นภาษาที่สามารถเรียนรู้ได้ง่าย รวดเร็ว และมีประสิทธิภาพในการทำงานสูง ไม่แพ้ภาษาระดับสูงอื่นๆ เช่น ภาษา C/C++ จาวา (Java) เพิร์ล (Perl) พีเอชพี (PHP) และวิชวลเบสิก เป็นต้น ซึ่งปัจจุบันภาษาไพธอนกำลังได้รับความนิยมเพิ่มขึ้นอย่างรวดเร็ว (จากผลการสำรวจของ CodeEval.com ในปี 2014 พบว่ามีผู้เขียนโปรแกรมด้วยภาษาไพธอนสูงที่สุดถึง 30.3% รองลงมาคือ Java 22.2%, C++ เท่ากับ 13%, Ruby 10.6%, JavaScript 5.2%, C# 5%, C 4.1%, PHP 3.3% ตามลำดับ) เห็นได้จากหน่วยงานที่มีชื่อเสียงระดับโลกนำไพธอนไปพัฒนาระบบงานของตนอย่างกว้างขวาง เช่น กูเกิล นาซ่า เป็นต้น ซึ่งจะกล่าวถึงในหัวข้อถัดไป สำหรับจุดเด่นของหนังสือเล่มนี้อยู่ตรงที่ ผู้เขียนได้อธิบายการเขียนโปรแกรมไพธอนเป็นลักษณะทีละขั้นตอนลำดับ (Step by Step) เริ่มตั้งแต่การเขียนโปรแกรมอย่างง่ายๆ ไปจนถึงการเขียนโปรแกรมขั้นสูง โดยมีภาพและโปรแกรมตัวอย่างประกอบคำอธิบาย ให้ผู้อ่านสามารถเข้าใจได้อย่างกว้างขวาง (เห็นตัวอย่างในมุมมองที่หลากหลาย) เนื้อหาครอบคลุมการใช้งานมากที่สุดจนถึงเวอร์ชันปัจจุบัน (เดือนมีนาคม 2557 คือเวอร์ชัน 3.4) รวมถึงเนื้อหาของหนังสือเล่มนี้ครอบคลุมในทุกๆ สายอาชีพ เช่น นักเรียน นิสิต นักศึกษา ครู อาจารย์ นักวิจัย นักวิทยาศาสตร์ นักคณิตศาสตร์ นักพัฒนาโปรแกรม งานฐานข้อมูล เว็บแอพพลิเคชัน ผู้ดูแลระบบคอมพิวเตอร์ และนักพัฒนาโปรแกรมระบบเครือข่าย เป็นต้น โครงสร้างที่สมควรอ่านหนังสือนี้เล่มนี้ ?

หนังสือเล่มนี้ได้ทำการกำหนดโครงสร้างของหนังสือออกเป็น 5 ภาค ประกอบไปด้วย

ภาคที่ 1 เป็นการแนะนำถึงการทำงานของระบบคอมพิวเตอร์พื้นฐาน และความสัมพันธ์กันอย่างใกล้ชิดระหว่างระบบคอมพิวเตอร์ฮาร์ดแวร์และโปรแกรม หรือซอฟต์แวร์ ที่ถูกใช้สำหรับควบคุมและสั่งงานให้ระบบคอมพิวเตอร์ฮาร์ดแวร์ สามารถทำงานได้ตรงตามเป้าหมาย และตรงความต้องการ

ภาคที่ 2 เป็นการอธิบายถึงพื้นฐานการพัฒนาโปรแกรม เช่น การเข้าใจชนิดของตัวแปรต่างๆ การกำหนดค่าตัวแปร คำสั่งการประมวลผลทางคณิตศาสตร์ โครงสร้างข้อมูล เงื่อนไข การทำซ้ำ พังชัน การจัดการข้อผิดพลาด อินพุต-เอาท์พุต การจัดการแฟ้มข้อมูล การตรวจสอบและค้นหาความผิดปกติของโปรแกรม และการเขียนโปรแกรมเชิงวัตถุ เป็นต้น

ภาคที่ 3 เป็นการแนะนำการเขียนโปรแกรมขั้นสูง เพื่อตอบสนองกับนักพัฒนาโปรแกรมที่มีความต้องการเฉพาะด้าน โดยยกตัวอย่างงานการเขียนโปรแกรมในมุมมองเชิงลึกกว่า ภาคที่ 2 เพื่อเป็นแนวทางให้ผู้สนใจสามารถนำทักษะ และตัวอย่างดังกล่าวไปประยุกต์กับงานของตนเองได้ เช่น นักพัฒนาโปรแกรมด้านระบบคอมพิวเตอร์ (System programming) นักวิเคราะห์ข้อมูลความและภาพ (Text and image processing) เว็บแอ��เพล็กชันและฐานข้อมูล (Web/web services and database developing) โปรแกรมด้านระบบเครือข่าย (Network programming) การโปรแกรมโพรเซสและเรด การโปรแกรมกับงานสารสนเทศ (GIS) โครงสร้างข้อมูล (Data Structure) รวมไปถึงการเขียนโปรแกรมภาษาไพธอนกับงานด้านวิทยาศาสตร์อื่นๆ อีกมากมาย

ภาคที่ 4 แนะนำการเขียนโปรแกรมกับกราฟิกด้วยภาษาไพธอน (GUI) เพื่ออำนวยความสะดวก สะดวกสำหรับนักพัฒนาโปรแกรมที่ต้องการกราฟิกเพื่อสนับสนุนการทำงาน เช่น ความรู้เบื้องต้นเกี่ยวกับงานด้านกราฟิก ไดอะล็อก เลเยอร์ การควบคุมอีเวนท์ เป็นต้น

ภาคที่ 5 แสดงไลบรารีพื้นฐานสำคัญๆ ของภาษาไพธอน เพื่อให้ผู้อ่านสามารถเลือกนำไปใช้กับงานได้อย่างเหมาะสม

ตารางต่อไปนี้ เป็นการสรุปว่าหนังสือเล่มนี้เหมาะกับใครบ้าง ดังนี้

ภาคที่	เหลือหาเหมาะสมกับใครบ้าง?
1	ผู้ที่สนใจ นักเรียน นิสิต นักศึกษา นักพัฒนาโปรแกรมระดับต้น ที่ไม่มีความรู้พื้นฐานด้านการพัฒนาโปรแกรมมาก่อน หรือเพิ่งเริ่มหัดเขียนโปรแกรม หรือไม่เคยศึกษาเรื่องการทำงานของระบบคอมพิวเตอร์ และซอฟต์แวร์คอมพิวเตอร์
2	ผู้ที่สนใจ นักเรียน นิสิต นักศึกษา อาจารย์ นักวิจัย นักพัฒนาโปรแกรมระดับต้น ถึงระดับกลาง ที่มีความรู้พื้นฐานด้านระบบคอมพิวเตอร์ และการพัฒนาโปรแกรมมาบ้างแล้ว หรือเคยเขียนโปรแกรมด้วยภาษาอื่นๆ มาแล้ว (แต่ถ้าต้องการปูพื้นฐานการทำงานของระบบคอมพิวเตอร์ ก็ควรอ่านภาคที่ 1 ประกอบด้วย)
3	ผู้ที่สนใจ นักเรียน นิสิต นักศึกษา อาจารย์ นักวิจัย นักพัฒนาโปรแกรมระดับกลาง ถึงสูง ที่มีความรู้พื้นฐานด้านระบบคอมพิวเตอร์ และการพัฒนาโปรแกรมมาเป็นอย่างดีแล้ว หรือเคยเขียนโปรแกรมด้วยภาษาอื่นๆ มาแล้ว หรือต้องการเปลี่ยนภาษา หรือต้องการใช้คุณสมบัติของภาษาไพธอนในการพัฒนางานให้สอดคล้องกับงานของตน หรือภาษาเดิมที่ใช้งานอยู่มีข้อจำกัดในการใช้งาน
4	ผู้ที่สนใจ นักเรียน นิสิต นักศึกษา อาจารย์ นักวิจัย นักพัฒนาโปรแกรมระดับเริ่มต้น ถึงสูง ที่มีความรู้พื้นฐานด้านระบบคอมพิวเตอร์ และการพัฒนาโปรแกรมมาแล้ว เพื่อตอบสนองต่องานด้านกราฟิก และตอบโต้กับผู้ใช้งานด้วยเมาร์ค เป็นต้น
5	สำหรับผู้ที่สนใจ โดยเขียนโปรแกรมภาษาไพธอนมาแล้ว

วิธีการอ่านหนังสือเล่มนี้

สำหรับวิธีการอ่านหนังสือเล่มนี้ ควรเริ่มจาก ตัวผู้อ่านต้องประเมินความสามารถของตนเองก่อน โดยการเทียบกับตารางด้านบนที่กล่าวมาแล้ว จากนั้นเริ่มอ่านตามความสามารถที่ผู้อ่านมีอยู่ ตัวอย่างเช่น เมื่อผู้อ่านประเมินตนเองแล้วว่าอยู่ในระดับเริ่มต้น (เคยเขียนโปรแกรมมาบ้างแต่เริ่มจะลืมเลือนแล้ว) เมื่อเทียบกับตารางแล้วผู้อ่านควรจะเริ่มต้นอ่านในภาคที่ 2 ซึ่งในภาคที่ 2 ก็มีหลายบท ประกอบอยู่ ในที่นี้ ถ้าผู้อ่านคิดว่าเข้าใจเรื่องเกี่ยวกับ นิพจน์ (Expression) เป็นอย่างดีแล้ว ก็สามารถข้ามไปอ่านในบทต่อไปของภาคที่ 2 ได้เลย แต่มีข้อที่ควรจดจำไว้ว่า แม้ว่าผู้อ่านจะเคยเขียนโปรแกรมด้วยภาษาอื่นมาเป็นอย่างดีแล้วก็ตาม แต่ไม่เคยได้ศึกษาภาษาไพธอนมาก่อนเลย ผู้เขียนแนะนำว่า ผู้อ่านควรอ่านภาคที่ 2 ทั้งหมด เนื่องจากไวยกรณ์ (Syntax) ของภาษาไพธอนจะมีความแตกต่างจากภาษาธรรมดากลุ่มนี้ พยายามทำความเข้าใจเพรียบเทียบระหว่างภาษาที่เคยใช้และภาษาไพธอน ให้ดัดแปลง และแก้ไขข้อด้อย ของภาษา ระดับสูงอีกครั้ง แล้วนำมารวบรวมไว้ในภาษาไพธอนนั่นเอง

สัญลักษณ์ที่ใช้กับหนังสือเล่มนี้

⇒	ต้องการเน้นว่าบรรทัดนั้นๆ ว่ามีความสำคัญ และควรอ่านอย่างละเอียด หรือหมายถึง ลำดับขั้นตอนของการทำงาน เช่น เปิดโปรแกรม ⇒ เลือกเมนู 1 ⇒ ... เป็นต้น
	Note: เนื้อหา ข้อความ หรือคำอธิบายที่ควรจดจำเอาไว้ เพราะจะเป็นประโยชน์ต่อการเขียนโปรแกรม
	Caution! เป็นสิ่งที่ผู้เขียนโปรแกรมควรระวัง หรือหลีกเลี่ยงการกระทำดังกล่าว
	Input file: เป็นแฟ้มข้อมูลอินพุตสำหรับใช้ในการเขียนโปรแกรม
	Tips: เป็นเคล็ดลับ ข้อเสนอแนะ หรือเกล็ดความรู้ที่ช่วยในการเขียนโปรแกรม
	OUTPUT: เอกสารพุทธของโปรแกรม
①, ②, ..	ลำดับขั้นตอนการทำงานของโปรแกรม

สารบัญโดยย่อ

หน้า

คำนำ

แนะนำเกี่ยวกับหนังสือ

ภาคที่ 1: แนะนำเบื้องต้นเกี่ยวกับการเขียนโปรแกรม.....1

บทที่ 1 ระบบคอมพิวเตอร์ทำงานอย่างไร.....2

บทที่ 2 โปรแกรมภาษาไทยตอน,18

ภาคที่ 2 : การเขียนโปรแกรมภาษาไทยตอนขั้นพื้นฐาน.....43

บทที่ 3 โครงสร้างการเขียนโปรแกรมภาษาไทยตอน.....45

บทที่ 4 ตัวแปร การกำหนดค่าและชนิดข้อมูล.....61

บทที่ 5 นิพจน์ ตัวดำเนินการ และตัวถูกดำเนินการ.....98

บทที่ 6 เงื่อนไข การตัดสินใจ การควบคุมทิศทาง และการทำซ้ำ.....111

บทที่ 7 พังชัน (Functions)153

บทที่ 8 โมดูลและแพ็คเกจ.....181

บทที่ 9 การจัดการข้อผิดพลาด.....190

บทที่ 10 การจัดการแฟ้มข้อมูล.....211

บทที่ 11 การเขียนโปรแกรมเชิงวัตถุ.....229

ภาคที่ 3: การเขียนโปรแกรมภาษาไทยตอนขั้นสูง.....269

บทที่ 12 นิพจน์ป กติ และการประมวลผลข้อความ.....270

บทที่ 13 ไฟรอนกับฐานข้อมูล.....304

บทที่ 14 เว็บและซีจีโอ.....317

บทที่ 15 การเขียนโปรแกรมกับมัลติชีรดและระบบเครือข่าย.....	343
บทที่ 16 การเขียนโปรแกรมระบบสารสนเทศภูมิศาสตร์เบื้องต้น.....	375
บทที่ 17 โครงสร้างข้อมูลเบื้องต้น.....	407
บทที่ 18 ปกิณกะโปรแกรมมิ่ง.....	433
ภาคที่ 4: การเขียนโปรแกรมกราฟฟิก.....	474
บทที่ 19 การเขียนโปรแกรมกราฟฟิกด้วย Python.....	475
ภาคที่ 5: Standard Library of Python.....	568
1. พังชันภาษาในสำหรับข้อมูลพื้นฐาน.....	568
2. พังชันภาษาในสำหรับข้อมูลเชิงประกอบ.....	597
3. เมธอดเกี่ยวกับการบริหารจัดการแฟ้มข้อมูลและไดเรคทรอรี่.....	613
บรรณานุกรม.....	631
ดัชนีคำศัพท์.....	634

สารบัญโดยละเอียด

หน้า

ภาคที่ 1: แนะนำเบื้องต้นเกี่ยวกับการเขียนโปรแกรม 1

บทที่ 1 ระบบคอมพิวเตอร์ทำงานอย่างไร 2

1. ความเข้าใจพื้นฐานเกี่ยวกับระบบคอมพิวเตอร์ 2
2. ภาษาคอมพิวเตอร์ 3
3. การแปลภาษา 6
4. ขั้นตอนการพัฒนาโปรแกรม 8
5. หลักการเขียนฟังก์ชัน 9
6. หลักการเขียนรหัสเทียม หรือชูโตโคด 14

บทที่ 2 โปรแกรมภาษาไพธอน 18

1. ภาษาไพธอนคืออะไร 18
2. คุณสมบัติ และความสามารถของภาษาไพธอน 18
3. ประวัติการสร้างภาษาไพธอน 22
4. งานด้านต่างๆ ที่ไพธอนสนับสนุนในปัจจุบัน 24
5. ตัวอย่างหน่วยงานที่เลือกใช้ไพธอน 26
6. ตัวอย่างซอฟต์แวร์ที่เขียนด้วยไพธอน 27
7. การติดตั้งและใช้งานไพธอนบนระบบปฏิบัติการวินโดวส์ 28
8. การติดตั้งและใช้งานไพธอนบนระบบปฏิบัติการยูนิกซ์-ลินุกซ์ 31
 - การติดตั้งแบบ package บนลินุกซ์ Centos/Fedora 31
 - การติดตั้งแบบ package บนลินุกซ์ Ubuntu/ LinuxMint/Debian 31
 - การติดตั้งแบบ Manual บนลินุกซ์-ยูนิกซ์ 32
9. การติดตั้ง และใช้งานไพธอนบนระบบปฏิบัติการแมคอินทอช 33
10. การใช้งาน Python shell (IDLE: Python GUI) และ Command line บนวินโดวส์ 33
 - วิธีการใช้งาน Python shell (IDLE: Python GUI) 33
 - วิธีการใช้งาน Python ผ่านทางคอมมานด์ไลน์ (Command line) 36
 - ปัญหาของ Python shell และ Python (Command line) 36
 - การส่งรันโปรแกรมพร้อมกับส่งค่าตัวแปร argv, argc 37
 - การส่งค่า Arguments ผ่านทาง Command-Line 38

การตรวจสอบความผิดพลาดของโปรแกรม.....	39
การตรวจสอบโปรแกรมโดยใช้ Debug.....	40
การตรวจสอบแบบใช้ Break point.....	41
สรุปการใช้งาน Python Debugger.....	42
ภาคที่ 2 : การเขียนโปรแกรมภาษาไพธอนขั้นพื้นฐาน.....	43
บทที่ 3 โครงสร้างการเขียนโปรแกรมภาษาไพธอน.....	45
1. สวัสดี ฉันคือไพธอน Hello, I'm Python.....	46
2. โครงสร้างการเขียนโปรแกรมภาษาไพธอน.....	48
3. ไวยกรณ์พื้นฐานที่จำเป็นอย่างยิ่งต้องจดจำ.....	49
4. คำสั่งการแสดงผล.....	52
การพิมพ์สตริง หรือสายอักขระ.....	54
การแปลงรูปแบบสายอักขระ.....	55
การแสดงผลเลขทศนิยม.....	57
การเติมสัญลักษณ์ และจัดตำแหน่งหน้าทศนิยม.....	58
5. คำสั่งรับค่าข้อมูลจากแป้นพิมพ์ หรือคีย์บอร์ด.....	58
6. คำสั่งช่วยเหลือ help ().....	60
บทที่ 4 ตัวแปร การกำหนดค่าและชนิดข้อมูล.....	61
1. หลักการตั้งชื่อตัวแปร (Identifier).....	61
2. การใช้งานตัวแปร (Using variables).....	62
3. คำสงวน (Reserved word, Keyword).....	64
4. ชนิดข้อมูล.....	64
ข้อมูลพื้นฐาน.....	64
ข้อมูลตัวเลข.....	64
ข้อมูลชนิดสายอักษร.....	69
ข้อมูลเชิงประกอบ.....	75
ลิสต์.....	75
ตัวดำเนินการพื้นฐานของลิสต์.....	77
ทัพเพล (Tuples).....	79
ตัวดำเนินการพื้นฐานที่ใช้กับทัพเพล.....	82

ดิกชันนารี (Dictionary).....	84
เซต (Sets).....	88
สรุปการใช้งานข้อมูลพื้นฐานและข้อมูลเชิงประจุบ.....	95
 บทที่ 5 นิพจน์ ตัวดำเนินการ และตัวถูกดำเนินการ.....	98
1. ตัวดำเนินการทางคณิตศาสตร์.....	98
2. ตัวดำเนินการทางด้านการเปรียบเทียบ.....	100
3. ตัวดำเนินการกำหนดค่า.....	101
4. ตัวดำเนินการระดับบิต.....	103
5. ตัวดำเนินการทางตรรกศาสตร์.....	106
6. ตัวดำเนินงานการเป็นสมาชิก.....	107
7. ตัวดำเนินการเอกลักษณ์.....	108
8. ลำดับความสำคัญของตัวดำเนินการ.....	109
 บทที่ 6 เรื่องไข การตัดสินใจ การควบคุมทิศทาง และการทำซ้ำ.....	111
1. การควบคุมทิศทางแบบเลือกทำ.....	112
การควบคุมทิศทางแบบ if.....	112
การควบคุมทิศทางแบบ if...else.....	115
การควบคุมทิศทางแบบ if...elif.....	118
การควบคุมทิศทางแบบ nested if.....	122
2. ควบคุมทิศทางแบบวนรอบ หรือทำซ้ำ (Loop, Iteration).....	126
คำสั่ง While loop.....	126
การวนซ้ำไม่รู้จบ (The Infinite Loop)	128
การใช้คำสั่ง else ร่วมกับ while และ for.....	129
คำสั่งควบคุมการทำซ้ำ (Loop control statements)	135
คำสั่ง break.....	136
คำสั่ง continue.....	138
คำสั่ง pass.....	139
คำสั่ง for loop.....	140
การอ้างถึงข้อมูลสมาชิกโดยการชี้ตำแหน่งของ for loop.....	142

พังชัน range.....	143
การใช้ else statement กับ for loop.....	145
ลูปซ้อน (Nested loops)	150
บทที่ 7 พังชัน (Functions)	153
1. พังก์ชันคืออะไร?.....	153
2. ประโยชน์ของพังชัน.....	153
3. การประกาศพังชัน.....	154
4. การเรียกใช้พังชัน.....	155
5. การส่งผ่านอาร์กิวเมนต์.....	156
Pass by value.....	157
Pass by reference.....	158
6. ชนิดของอาร์กิวเมนต์ที่ส่งให้พังชัน.....	160
Required arguments.....	160
Keyword arguments.....	161
Default arguments.....	162
Variable-length arguments.....	163
7. การสร้างพังก์ชันโดยไม่ระบุชื่อ.....	164
ข้อดีของ Anonymous functions.....	165
ข้อเสียของ Anonymous functions.....	165
8. การส่งค่ากลับจากพังชัน.....	169
9. การส่งค่ากลับจากพังชันหลายค่า.....	173
10. ขอบเขตของตัวแปร.....	174
ตัวแปรโกลบลอล.....	174
ตัวแปรชนิดโลคอล.....	174
11. การเรียกด้วยเงื่อน หรือการเวียนเกิด.....	177
บทที่ 8 โมดูลและแพ็คเกจ.....	181
1. โมดูลคืออะไร?	181
2. การเรียกใช้งานโมดูล.....	182

การใช้คำสั่ง import.....	182
การใช้คำสั่ง from module import function.....	182
การใช้คำสั่ง from module import *.....	182
การใช้คำสั่ง import OldModule as NewModule.....	183
ความแตกต่างระหว่าง import, from module import function และ import *.....	185
ตำแหน่งที่อยู่ของการโหลดโมดูลมาใช้งาน.....	186
PYTHONPATH.....	187
การโหลดโมดูลมาใช้งานใหม่.....	187
3. แพ็คเกจ (Packages)	187
 บทที่ 9 การจัดการข้อผิดพลาด.....	190
1. ข้อผิดพลาด (Exceptions) คืออะไร?.....	190
2. การจัดการข้อผิดพลาด (Exceptions handling)	191
การตรวจจับความผิดพลาดแบบไม่กำหนด Exceptions.....	194
การตรวจจับความผิดพลาดแบบหลาย Exceptions.....	195
การตรวจจับความผิดพลาดแบบ try...else...finally.....	197
อาร์กิวเมนต์ของ Exception.....	200
การตรวจจับความผิดพลาดด้วยคำสั่ง Raising.....	202
การสร้าง Exception ขึ้นมาใช้งานเอง.....	205
3. การยืนยันในสมมติฐาน (Assertions)	206
ข้อกำหนดในการใช้ assert.....	206
ประโยชน์ของ assert.....	207
ตำแหน่งที่นิยมวาง assert ไว้.....	207
คำแนะนำเพิ่มเติมอื่นๆ เกี่ยวกับการใช้งาน assert.....	207
 บทที่ 10 การจัดการแฟ้มข้อมูล.....	211
1. แฟ้มข้อมูลคืออะไร?	211
2. การบริหารจัดการกับแฟ้มข้อมูล.....	211
การเปิดและปิดแฟ้ม.....	212
การเปิดแฟ้ม.....	212

คุณสมบัติที่เกี่ยวข้องกับแฟ้มข้อมูล	214
การปิดแฟ้ม	214
การดำเนินการกับแฟ้มข้อมูล	216
การอ่านแฟ้มข้อมูล	216
การอ่านแฟ้มข้อมูลที่ละเอียด	219
การอ่านแฟ้มข้อมูลที่ละเอียดคำ	219
ตัวชี้ตำแหน่งในแฟ้มข้อมูล	221
การเขียนแฟ้มข้อมูล	223
การใช้คำสั่ง with กับแฟ้มข้อมูล	224
การจัดการแฟ้มและไดเรคทรอรี	225
บทที่ 11 การเขียนโปรแกรมเชิงวัตถุ.....	229
1. แนวคิดเกี่ยวกับหลักการเขียนโปรแกรม	229
2. แนวคิดเกี่ยวกับการโปรแกรมเชิงวัตถุ	229
แนวคิดการเขียนโปรแกรมเชิงพังก์ชัน	230
แนวคิดเชิงวัตถุ (Object)	230
ข้อดี-ข้อเสียของการเขียนโปรแกรมเชิงวัตถุ	231
ความหมายการโปรแกรมเชิงวัตถุ	231
3. เริ่มต้นเขียนโปรแกรมเชิงวัตถุด้วย Python	237
การสร้างคลาส	237
การสร้างอินสแตนซ์ของวัตถุ	244
การเข้าถึงแอตทริบิวต์และเมธอด	246
แอตทริบิวต์ชนิด built-in ในคลาส	247
การลบวัตถุที่สร้างขึ้น	249
การสืบทอดคุณสมบัติของคลาส	251
การโอเวอร์ไรร์ดิ้งเมธอด	258
การโอเวอร์โหลดดิ้งเมธอด	261
การโอเวอร์โหลดตัวดำเนินการ	263
การห่อหุ้มข้อมูล/การซ่อนข้อมูล	266
ภาคที่ 3: การเขียนโปรแกรมภาษาไพธอนขั้นสูง.....	269

บทที่ 12 นิพจน์ป กติ และการประมวลผลข้อความ.....270

1. นิพจน์ป กติคืออะไร?.....	270
2. นิพจน์ป กติอย่างง่าย.....	271
3. นิพจน์ป กติใน Python.....	274
4. ไวยกรรมของนิพจน์ป กติ.....	274
1. สัญลักษณ์ที่ใช้แทนตัวอักษรใดๆ เพียง 1 ตัว (.).....	275
2. สัญลักษณ์ที่ใช้เลือกตัวอักษรใดๆ เพียง 1 ตัว จากกลุ่มของอักษร ([...]).....	276
3. สัญลักษณ์พิเศษอื่นๆ ที่ใช้สร้าง Regex.....	278
4. การค้นหาคำที่ทำແเน່ງເຮີມຕົນແລະສິ້ນສຸດຂອງຂໍ້ວາມ.....	289
5. เมธอด re.search().....	291
6. การค้นหาและແທນທີ່ຄຳດ້ວຍເມຫວດ re.sub().....	297
7. ເມຫວດ re.findall().....	297
8. ເມຫວດ re.match().....	299
9. ເມຫວດ re.split().....	300
10. ການກຳຫັດຊື່ໃຫ້ກັບ regex (Named group)	301
11. ຕົວຢ່າງການເຂົ້ານ regex ພຣັນຄໍາອົບປາຍ.....	302

บทที่ 13 ໂພອນກັບຮ້ານຂໍ້ອມ.....304

1. ຮ້ານຂໍ້ອມຈຶ່ງໃຫ້ໄດ້.....	304
2. ການໃໝ່ໂພອນກັບຮ້ານຂໍ້ອມ.....	305
3. ການສ້າງຮ້ານຂໍ້ອມແລະຕາரາງ.....	309
4. ຄຳສັ່ງຕໍ່ເນີນການເກື່ອກກັບຮ້ານຂໍ້ອມ.....	311
1. ຄຳສັ່ງການເພີ່ມຮະເບີ່ນ (Insert Operation).....	311
2. ຄຳສັ່ງການອ່ານຮະເບີ່ນ (Read Operation).....	312
3. ຄຳສັ່ງການປັບປຸງຮະເບີ່ນ (Update Operation).....	313
4. ຄຳສັ່ງການລົບຮະເບີ່ນ (Delete Operation).....	314
5. ການຍື່ນຍັນແລະຍົກເລີກຄຳສັ່ງການດໍາເນີນການ.....	315
6. ຄຳສັ່ງຍົກເລີກການເຊື່ອມຕ່ອກກັບຮ້ານຂໍ້ອມ (Disconnect).....	316

บทที่ 14 เว็บและชีวิโไอ.....	317
1. เว็บและชีวิโไอคืออะไร?.....	317
2. การติดตั้งเซิร์ฟเวอร์และการปรับแต่ง.....	320
3. การเขียนโปรแกรม CGI	322
เมธอด HTTP GET.....	326
เมธอด HTTP POST.....	326
การส่งข้อมูลของ Checkbox ไปประมวลผลกับ CGI ศคริปต์.....	329
การส่งข้อมูลของ Radio button ไปประมวลผลกับ CGI ศคริปต์.....	330
การส่งข้อมูลของ Text Area ไปประมวลผลกับ CGI ศคริปต์.....	331
การส่งข้อมูลของ Drop Down Box ไปประมวลผลกับ CGI ศคริปต์.....	333
การใช้คุกเก้ (Cookie) ใน CGI ศคริปต์.....	334
การอัพโหลดแฟ้มด้วย CGI ศคริปต์.....	337
การประยุกต์ CGI ศคริปต์ทำงานร่วมกับฐานข้อมูล.....	339
บทที่ 15 การเขียนโปรแกรมกับมัลติเทรดและระบบเครือข่าย.....	343
1. เทรดและforex.....	343
2. การเขียนโปรแกรมกับเทรด.....	345
1. ไม่ดูด Threading.....	346
2. ซิงโครไนซ์เทรด (Threads Synchronization).....	349
3. การจัดลำดับความสำคัญของคิวสำหรับมัลติเทรด.....	352
3. การเขียนโปรแกรมเครือข่ายแบบ-client-server (Clients-Servers).....	355
1. หมายเลขไอพีแอดเดรส (IP Address).....	355
2. พอร์ต, เซิร์ฟเวอร์ และclient (Ports, Servers and Clients).....	356
3. เริ่มต้นการเขียนโปรแกรมเซิร์ฟเวอร์-client.....	357
4. การเขียนโปรแกรมพูดคุยผ่านเน็ตเวิร์ค (Chat).....	363
5. โปรแกรม Chat ที่สามารถรองรับclient พร้อมๆ กันหลายเครื่อง.....	367
บทที่ 16 การเขียนโปรแกรมระบบสารสนเทศภูมิศาสตร์เบื้องต้น.....	375
1. ระบบสารสนเทศภูมิศาสตร์.....	375

2. การติดตั้งโปรแกรม ArcGIS.....	377
3. การเขียนโปรแกรมไฟชอนกับ GIS (ArcGIS 10.x)	379
3.1 การเรียกใช้งาน Arcpy site package ด้วยไฟชอน.....	379
3.2 การเรียกใช้ Tools ใน ArcGIS ด้วยไฟชอน.....	381
3.3 การสร้างตัวแปรเพื่อใช้งานร่วมกับ arcpy.....	383
3.4 การเรียกใช้งานโมดูลใน arcpy package.....	385
3.5 การอ้างอิงแพนที่ที่ใช้งานภายใน ArcMap.....	385
3.6 การอ้างอิงแพนที่ที่ใช้งานภายนอก ArcMap.....	386
3.7 การแสดงเฟรมข้อมูล (Data frame).....	387
3.8 การแสดงレイเยอร์ (Layers).....	388
3.9 การคัดกรองเฉพาะレイเยอร์ที่ต้องการ.....	389
3.10 การปรับเปลี่ยนขอบเขตของแพนที่.....	390
3.11 แสดงรายชื่อตาราง (Tables)	391
3.12 การเพิ่มเลเยอร์ในเอกสารแพนที่ โดยใช้ AUTO_ARRANGE.....	392
3.13 การเพิ่มเลเยอร์ลงในแพนที่โดยการกำหนดตำแหน่งในเฟรมข้อมูล	393
3.14 การเปลี่ยนสัญลักษณ์ (Symbology) ในレイเยอร์.....	394
3.15 ปรับปรุงคุณสมบัติต่างๆ ในレイเยอร์ (Layer properties)	395
3.16 การค้นหาตำแหน่งข้อมูลต้นฉบับของแฟ้มแพนที่และแฟ้มレイเยอร์.....	398
3.17 การแก้ปัญหาแหล่งข้อมูลผิดพลาดด้วยเมธอด	400
3.18 การแก้ไขแหล่งข้อมูลด้วยคำสั่ง MapDocument.replaceWorkspaces().....	401
3.19 แก้ไขแหล่งข้อมูลในระดับレイเยอร์ (Layer) และตาราง (TableView).....	403
3.20 การค้นหาการเชื่อมโยงแหล่งข้อมูลที่ผิดพลาดทั้งหมดในไฟล์เรคอร์ด.....	405
 บทที่ 17 โครงสร้างข้อมูลเบื้องต้น.....	407
1. การประเมินประสิทธิภาพอัลกอริทึมจากเวลาที่ใช้ทำงาน.....	407
2. การประเมินประสิทธิภาพอัลกอริทึมจากการนับจำนวนบรรทัดคำสั่ง.....	409
3. การประเมินประสิทธิภาพอัลกอริทึมจากการคำนวณหน่วยความจำ.....	411
4. การประเมินประสิทธิภาพอัลกอริทึมด้วยพังก์ชันอัตราการเติบโต	413
5. การวิเคราะห์ Best-case, Worst-case และ Average-case.....	419
6. การจัดเรียงข้อมูล (Sorting)	419
1. การจัดเรียงข้อมูลแบบ Insertion Sort.....	420

2. การจัดเรียงข้อมูลแบบ Selection Sort.....	422
3. การจัดเรียงข้อมูลแบบ Bubble Sort.....	425
4. การจัดเรียงข้อมูลแบบ Merge Sort.....	427
7. การค้นหาข้อมูล (Searching)	429
1. การค้นหาข้อมูลแบบลำดับ (Sequential Search)	429
2. การค้นหาข้อมูลแบบพับครึ่ง (Binary Search)	431

บทที่ 18 ปกิณกะโปรแกรมมิ่ง.....433

1. วันและเวลา (Date/Time).....	433
2. ส่งจดหมายอิเล็กทรอนิกส์ (Email)	435
3. การประยุกต์ใช้ฟังชัน map, filter, reduce, lambda กับข้อมูลแบบลำดับ.....	437
4. ไฟร่อน Generators, yield.....	441
5. การเขียนโปรแกรมไฟร่อนด้วย Eclipse.....	443
6. การแปลงไฟรอนสคริปต์เป็น EXE (executable)	451
7. Decorator Vs Function.....	453
8. เมธอด .format().....	457
9. อาเรย์และเมทริกซ์ (Array and Matrix)	459
10. การสร้างเน็ตเวิร์คกราฟด้วย NetworkX.....	464
11. การพรีอตกราฟ (Plot Graph) ด้วย MATPLOTLIB, NUMPY และ SCIPY.....	468

ภาคที่ 4: การเขียนโปรแกรมกราฟฟิก.....472

บทที่ 19 การเขียนโปรแกรมกราฟฟิกด้วยไฟร่อน.....473

1. แนวความคิดในการออกแบบ GUI.....	474
2. การสร้าง GUI ด้วย Tk.....	478
1. Widgets.....	478
2. การเรียกใช้งาน Widgets.....	479
3. การจัดการรูปทรงเลขานุณิตให้กับ Widgets (Geometry management).....	481
4. การจัดการกับเหตุการณ์ต่างๆ (Event Handling).....	482
5. การตอบสนองต่อเหตุการณ์ที่เกิดขึ้น (Command Callbacks).....	482

6. การผูกเหตุการณ์ต่างๆ เข้ากับไฟชอน.....	483
7. สรุปขั้นตอนการสร้าง GUI ด้วย tkinter และ tkk.....	484
3. การจัดวาง Widgets ด้วยเลขากนิต (Geometry management).....	485
1. เมธอด pack().....	485
2. เมธอด grid().....	487
3. เมธอด place().....	488
4. คุณสมบัติพื้นฐานของ Widgets.....	489
1. Dimension.....	489
2. Color.....	490
3. Font.....	491
4. Anchors.....	492
5. Relief styles.....	492
6. Bitmaps.....	493
7. Cursors.....	494
5. การสร้างและใช้งาน Widgets พื้นฐาน.....	487
1. Frame.....	496
2. Button.....	499
3. Canvas.....	502
4. Checkbutton.....	506
5. Entry.....	510
6. Label.....	514
7. Listbox.....	517
8. Menubutton.....	522
9. Message.....	524
10. Menu.....	525
11. Radiobutton.....	530
12. Scale.....	532
13. Scrollbar.....	535
14. Text.....	538
15. Toplevel.....	543
16. Spinbox.....	545
17. LabelFrame.....	547

18. MessageBox.....	449
19. Widgets อื่นๆ ที่นำเสนอไป.....	550
6. ตัวอย่างการประยุกต์ใช้งานใน GUI.....	552
1. เครื่องคิดเลขขนาดเล็ก (Mini-Calculator).....	552
2. เกมส์ Tic-Tac-Toe.....	557
ภาคที่ 5: Standard Library of Python.....	567
1. พังชันภายในสำหรับข้อมูลพื้นฐาน.....	568
การเปลี่ยนชนิดข้อมูล.....	568
พังชันคำนวณทางคณิตศาสตร์.....	572
พังชันสูตรตัวเลข.....	576
พังชันตรีโกณมิติ.....	578
ค่าคงที่ทางคณิตศาสตร์.....	582
เมธอดและพังชันที่ดำเนินการกับสตริง.....	582
2. พังชันภายในสำหรับข้อมูลเชิงประกอบ.....	597
พังชัน และเมธอดสำหรับลิสต์.....	598
พังชันสำหรับลิสต์.....	598
เมธอดสำหรับลิสต์.....	600
พังชันภายในสำหรับทัพเพิล.....	603
พังชันภายในสำหรับทัพเพิล.....	603
พังชัน และเมธอดสำหรับดิกชันนารี.....	604
พังชันสำหรับดิกชันนารี.....	604
เมธอดสำหรับดิกชันนารี.....	605
พังชัน และเมธอดสำหรับเซต.....	609
พังชันสำหรับเซต.....	609
เมธอดสำหรับเซต.....	609
3. เมธอดเกี่ยวกับการบริหารจัดการแฟ้มข้อมูลและไดเรคทรอรี่.....	613
บรรณานุกรม.....	631
ดัชนีคำศัพท์.....	634

ภาคที่ 1

แนะนำเบื้องต้นเกี่ยวกับการเขียนโปรแกรม
(Introduction to basics programming)

บทที่ 1 ระบบคอมพิวเตอร์ทำงานอย่างไร

บทที่ 2 โปรแกรมภาษาไพธอน

บทที่ 1

ระบบคอมพิวเตอร์ทำงานอย่างไร

(How does the computer system work?)



1. ความเข้าใจพื้นฐานเกี่ยวกับระบบคอมพิวเตอร์ (Basics of computer system)

ระบบคอมพิวเตอร์มีความสามารถในการคำนวณ ประมวลผลข้อมูลด้วยความเร็วสูง และลดภาระงานอันซับซ้อนที่มนุษย์ทำได้ยากให้ลดลง ระบบคอมพิวเตอร์ประกอบไปด้วย 5 ส่วนที่สำคัญ คือ



- 1) อุปกรณ์ฮาร์ดแวร์ (Hardware) หมายถึง อุปกรณ์ต่างๆ ที่ประกอบเข้าเป็นตัวเครื่องคอมพิวเตอร์ เช่น หน่วยประมวลผลกลาง (CPU : Central processing unit) มีหน้าที่ในการประมวลผลคำสั่ง หรือข้อมูลต่างๆ หน่วยความจำหลัก (Main memory) ทำหน้าที่จัดเก็บข้อมูลหรือคำสั่งต่างๆ เพื่อประมวลผล อุปกรณ์แสดงข้อมูล (Output device) ทำหน้าที่ในการแสดงผลต่างๆ ได้แก่ จอภาพ และเครื่องพิมพ์ หน่วยความจำสำรอง (Auxiliary storage) มีหน้าที่ในการจัดเก็บข้อมูลชนิดถาวร เช่น ฮาร์ดดิสก์ อุปกรณ์รับข้อมูล (Input device) เช่น เม้าส์ คีย์บอร์ด และตัวถังหรือตัวกล่องคอมพิวเตอร์ (Case) เป็นต้น

- 2) ชุดของคำสั่งที่ทำหน้าที่สั่งงาน และควบคุมการทำงานของคอมพิวเตอร์ เรียกว่า โปรแกรม หรือซอฟต์แวร์คอมพิวเตอร์ (Computer programs) ซอฟต์แวร์ดังกล่าวถูกพัฒนามาจากภาษาคอมพิวเตอร์ (Programming language) ภาษาใดภาษาหนึ่ง เช่น ภาษา C/C++ เป็นต้น โดยโปรแกรมเมอร์ (Programmer) ซึ่งเป็นผู้พัฒนาซอฟต์แวร์ดังกล่าว สำหรับซอฟต์แวร์คอมพิวเตอร์ แบ่งออกเป็น 2 ประเภทคือ

- ซอฟต์แวร์ระบบ (System software) เป็นซอฟต์แวร์ที่ทำหน้าที่จัดการ และควบคุม ทรัพยากร่างๆ ของเครื่องคอมพิวเตอร์ และอำนวยความสะดวกด้านเครื่องมือสำหรับการทำงานพื้นฐานต่างๆ ตั้งแต่ผู้ใช้เริ่มเปิดเครื่องคอมพิวเตอร์ไปจนถึงการปิดระบบ เช่น ระบบปฏิบัติการ (Operating system) และซอฟต์แวร์ที่ติดต่อ และควบคุมอุปกรณ์ฮาร์ดแวร์โดยตรง (Device driver program) เป็นต้น

- ซอฟต์แวร์ประยุกต์ (Application software) หมายถึง ซอฟต์แวร์ที่พัฒนาขึ้น เพื่อใช้งานเฉพาะด้านตามความต้องการของผู้ใช้ เช่น งานด้านการจัดทำเอกสาร การทำบัญชี ระบบฐานข้อมูล หรือเว็บแอปพลิเคชัน เป็นต้น

3) ข้อมูล/สารสนเทศ (Data/Information) คือ ข้อมูลต่างๆ ที่นำมาประมวลผลด้วยระบบ



คอมพิวเตอร์ เพื่อให้ได้ผลลัพธ์ตามที่ผู้ใช้ต้องการ ยกตัวอย่างข้อมูลเกี่ยวกับนักศึกษา เช่น รหัสนักศึกษา ชื่อ-นามสกุล ที่อยู่ หมายเลขโทรศัพท์ ผลการศึกษา ที่อยู่อาศัย เป็นต้น

4) บุคคลากร (Peopleware) คือ เจ้าหน้าที่ปฏิบัติงานต่างๆ และผู้ใช้เครื่องคอมพิวเตอร์ในหน่วยงานนั้นๆ บุคลากรด้านคอมพิวเตอร์มีความสำคัญมาก เพราะการใช้เครื่องคอมพิวเตอร์ทำงานต่างๆ นั้นจะต้องมีการจัดเตรียมระบบก่อน



เสมอ

5) กระบวนการทำงาน (Documentation/Procedure) เป็นขั้นตอนการทำงานเพื่อให้ได้



ผลลัพธ์จากคอมพิวเตอร์ ในการทำงานกับคอมพิวเตอร์จำเป็นต้องให้ผู้ใช้เข้าใจขั้นตอนการทำงาน ต้องมีระเบียบปฏิบัติให้เป็นแบบเดียวกัน มีการจัดทำคู่มือ การบำรุงรักษาเครื่องคอมพิวเตอร์ และซอฟต์แวร์ ให้เป็นมาตรฐานเดียวกัน

2. ภาษาคอมพิวเตอร์ (Computer languages)

ภาษาคอมพิวเตอร์ คือ ภาษาที่มนุษย์สร้างขึ้นเพื่อใช้ติดต่อสื่อสารกับระบบคอมพิวเตอร์ เพื่อให้สามารถปฏิบัติตามคำสั่งได้ โดยแบ่งออกเป็น 5 ระดับ คือ

1) ภาษาเครื่อง (Machine language) เป็นภาษาที่เกิดขึ้นในยุคแรกสุด และเป็นภาษาเดียวที่เครื่องคอมพิวเตอร์สามารถทำความเข้าใจคำสั่งได้ ภาษาเครื่องจะแทนข้อมูล หรือคำสั่งในโปรแกรมด้วยกลุ่มของตัวเลข 0 และ 1 หรือที่เรียกว่าเลขฐานสอง ซึ่งจะสัมพันธ์กับการเปิด (On) และการปิด (Off) ของสัญญาณไฟฟ้าภายในเครื่องคอมพิวเตอร์ ดังนั้นจึงเป็นภาษาที่มีการทำงานที่รวดเร็วที่สุด ตารางที่ 1.1 แสดงตัวอย่างคำสั่งของภาษาเครื่อง

ตารางที่ 1.1 แสดงคำสั่งภาษาเครื่อง

ภาษาเครื่อง	การดำเนินการ	ความหมาย
1000000100100101	Load Memory 5 -> R1	โหลดข้อมูลในหน่วยความจำตำแหน่งที่ 5 ให้กับรีสเตอร์ R1
1000000101000101	Load Memory 10 -> R2	โหลดข้อมูลในหน่วยความจำตำแหน่งที่ 10 ให้กับรีสเตอร์ R2
1010000100000110	R1 + R2 -> R0	รวมค่า R1 และ R2 แล้วเก็บในรีสเตอร์ R0

1000001000000110	Store R0 -> Memory 6	นำข้อมูลใน R0 เก็บลงหน่วยความจำตำแหน่งที่ 6
1111111111111111	HALT	หยุดการทำงาน

- 2) ภาษาระดับต่ำ (Low level language) สืบเนื่องจากภาษาเครื่องเป็นภาษาที่เข้าใจได้ยาก และมีความยุ่งยากเป็นอย่างมากในการเขียนโปรแกรม ดังนั้นจึงไม่เป็นที่นิยม ส่งผลให้มีการใช้งานอย่างจำกัด ดังนั้นจึงได้มีผู้คิดค้นพัฒนาภาษาคอมพิวเตอร์ขึ้นใหม่เพื่อแก้ปัญหาดังกล่าว เรียกว่า ภาษาแอสเซมบลี (Assembly language) โดยการนำตัวอักษรภาษาอังกฤษเป็นรหัสแทนการทำงาน และตั้งชื่อตัวแปรแทนตำแหน่งที่อยู่ที่ใช้เก็บข้อมูล การใช้สัญลักษณ์ช่วยให้การเขียนโปรแกรมทำความเข้าใจได้ง่ายยิ่งขึ้น และยังคงคุณสมบัติด้านความเร็วเหมือนกับภาษาระดับต่ำไว้ด้วย ตัวอย่างในตารางที่ 1.2
- ตารางที่ 1.2 แสดงคำสั่งภาษาระดับต่ำ (Assembly code)**

ภาษาแอสเซมบลี	การดำเนินการ	ความหมาย
LOAD R1, M[5]	Load Memory 5 -> R1	โหลดข้อมูลในหน่วยความจำตำแหน่งที่ 5 ให้กับรีสเตอร์ R1
LOAD R2, M[10]	Load Memory 10 -> R2	โหลดข้อมูลในหน่วยความจำตำแหน่งที่ 10 ให้กับรีสเตอร์ R2
ADD R0, R1, R2	R1 + R2 -> R0	รวมค่า R1 และ R2 แล้วเก็บในรีสเตอร์ R0
SAVE R0, M[6]	Store R0 -> Memory 6	นำข้อมูลใน R0 เก็บลงหน่วยความจำตำแหน่งที่ 6
HALT	HALT	หยุดการทำงาน

โปรแกรมที่ถูกเขียนด้วยภาษาแอสเซมบลีนี้ ยังไม่สามารถนำไปใช้งานกับเครื่องคอมพิวเตอร์ได้ทันที จำเป็นต้องมีการแปลคำสั่งภาษาแอสเซมบลีเป็นภาษาเครื่องเสียก่อน สำหรับโปรแกรมแปลภาษาที่มีชื่อว่า “แอสเซมเบลอร์ (Assembler)” ซึ่งแตกต่างไปตามสถาปัตยกรรมของเครื่องคอมพิวเตอร์แต่ละชนิด ดังนั้นแอสเซมเบลอร์จะไม่สามารถใช้แปลโปรแกรมภาษาแอสเซมบลีข้ามสถาปัตยกรรมได้ เช่น สถาปัตยกรรม 16 บิต กับ 32 บิต เป็นต้น แม้ว่าภาษาแอสเซมบลีใช้ตัวอักษรช่วยในการพัฒนาโปรแกรมแล้วก็ตาม แต่ก็ยังมีความห่างไกลกับภาษาที่มนุษย์สามารถเข้าใจได้อยู่มาก ภาษาแอสเซมบลีจึงมีผู้ใช้งานเพิ่มขึ้นไม่มากนัก โดยส่วนใหญ่มักนิยมใช้ในการณ์ที่ต้องการควบคุมการทำงานของเครื่องคอมพิวเตอร์โดยตรง

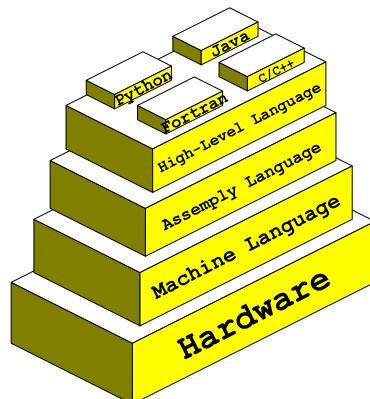
- 3) ภาษาระดับสูง (High level language) เป็นภาษาที่ถูกสร้างขึ้นเพื่อช่วยอำนวยความสะดวกในการเขียนโปรแกรม กล่าวคือ ลักษณะของคำสั่งจะประกอบด้วยคำต่างๆ ในภาษาอังกฤษ ซึ่งผู้อ่านสามารถทำความเข้าใจความหมายได้ทันที จึงทำให้ผู้เขียนโปรแกรมสามารถเพิ่มขีดความสามารถในการพัฒนาโปรแกรมได้ดีกว่าการเขียนโปรแกรมด้วย

ภาษาระดับต่ำอย่างภาษาแอสแซมบลี หรือภาษาเครื่อง สำหรับตัวอย่างโปรแกรมภาษา
ระดับสูง อาทิเช่น ภาษาฟอร์TRAN (FORTRAN) ภาษาโอบอล (COBOL) ภาษาปาสคัล
(Pascal) ภาษาเบสิก (BASIC) ภาษาวิชลเบสิก (Visual Basic) ภาษาซี (C) และภาษาจาวา
(Java) เป็นต้น โปรแกรมที่เขียนด้วยภาษาระดับสูง แต่ละภาษาจะต้องมีโปรแกรมที่ทำ
หน้าที่แปลภาษาระดับสูงให้เป็นภาษาเครื่องเสมอ เช่น โปรแกรมแปลภาษา C/C++ เป็น<sup>ภาษาเครื่อง โปรแกรมแปลภาษาปาสคัลเป็นภาษาเครื่อง เป็นต้น โดยปกติแล้ว คำสั่งแต่
ละคำสั่งในภาษาระดับสูงจะถูกแปลเป็นภาษาเครื่องมากกว่า 1 คำสั่งเสมอ ดังตัวอย่างใน
ตารางที่ 1.3</sup>

ตารางที่ 1.3 แสดงการเปรียบเทียบระหว่างคำสั่งภาษาระดับสูง (ภาษา C)/ต่ำ/เครื่อง

ภาษาระดับสูง	ภาษาแอสแซมบลี	ภาษาเครื่อง
int X=880;	X: .word 880	
X = X + 5;	LOAD R1 X ADD R2 R1 #5 STORE R2 X	1001001100001111000000000000100000 10011001000011110000000000000000000000 100000111110001000000000000000000000000 10110011000100100000000000000000101 1001001100001111000000000000100000 10011001000011110000000000000000000000 1000010111100100000000000000000000000000 111
	HALT	

รูปที่ 1.1 แสดงความสัมพันธ์ระหว่างคอมพิวเตอร์ฮาร์ดแวร์ ภาษาเครื่อง ภาษาแอสแซมบลี
และ ภาษาระดับสูง สังเกตว่าภาษาที่มีความเร็วสูงที่สุดจะอยู่ในระดับต่ำที่สุด ภาษาที่มี
มนุษย์เข้าใจได้ง่ายที่สุดจะอยู่ในระดับบนสุด (ใกล้เคียงภาษามนุษย์มากที่สุด) และภาษาที่
คอมพิวเตอร์เข้าใจได้ดีที่สุดคือภาษาเครื่องนั้นเอง (อยู่ในระดับต่ำที่สุด)



รูปที่ 1.1 แสดงความสัมพันธ์ระหว่างภาษาคอมพิวเตอร์ชนิดต่างๆ กับฮาร์ดแวร์

- 4) ภาษาระดับสูงมาก (Very high-level language) เป็นภาษาที่ผู้เขียนโปรแกรมเพียงแต่กำหนดว่าต้องการให้โปรแกรมทำอะไรบ้างก็สามารถเขียนโปรแกรมได้ทันที โดยไม่ต้องทราบว่าทำงานได้อย่างไร ทำให้การเขียนโปรแกรมสามารถทำได้ง่าย และรวดเร็ว ข้อมูลจะเก็บอยู่ในฐานข้อมูล (Database) เรียกภาษานี้ว่า ภาษาเรียกค้นข้อมูล (Query language) หรือ SQL (Structured Query Language) ตัวอย่างของภาษาแสดงในตารางที่ 1.4

ตารางที่ 1.4 แสดงคำสั่งในภาษา SQL

คำสั่งในภาษา SQL	ความหมาย
<pre>select * from employees where employee_id > 1 order by employee_id ASC</pre>	ดึงข้อมูลจาก Table employees โดยต้องการดึง แค่ที่มี employee_id มากกว่า 1 และเรียงตาม employee_id จากน้อยไปมาก

- 5) ภาษาธรรมชาติ (Natural languages) เรียกได้ว่าเป็น ภาษาธรรมชาติ เนื่องจากมีความใกล้เคียงกับภาษามนุษย์มากที่สุด ภาษาที่สามารถรองรับเทคโนโลยีปัญญาประดิษฐ์ (Artificial Intelligence : AI) หรือ ระบบผู้เชี่ยวชาญ (Expert System) ซึ่งทำให้เครื่องคอมพิวเตอร์สามารถที่จะเข้าใจคำสั่งจากเสียงพูด และโต้ตอบกับมนุษย์ได้ เช่น ภาษาโปรล็อก (Prolog) ภาษาลิสป (Lisp) เป็นต้น จากตารางที่ 1.5 เป็นตัวอย่างการทำงานของภาษาโปรล็อก ที่ต้องการรวมค่าตัวเลข 2 จำนวนเข้าด้วยกัน

ตาราง 1.5 ตัวอย่างภาษาธรรมชาติ (Prolog)

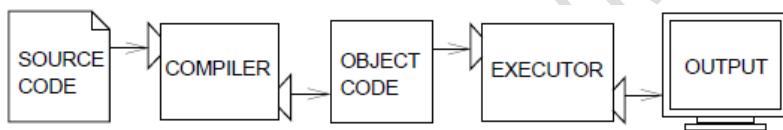
คำสั่งภาษาโปรล็อก	ผลการทำงาน
<pre>tart:- sum,nl. sum:- write('X= '), read(X), write('Y= '), read(Y), S is X+Y, write('Sum is '), write(S).</pre>	<pre>?- start. X= : 1. Y= : 2. Sum is 3 yes</pre>

3. การแปลภาษา (Translation)

การแปลภาษาของคอมพิวเตอร์แบ่งออกเป็น 2 ประเภทคือ

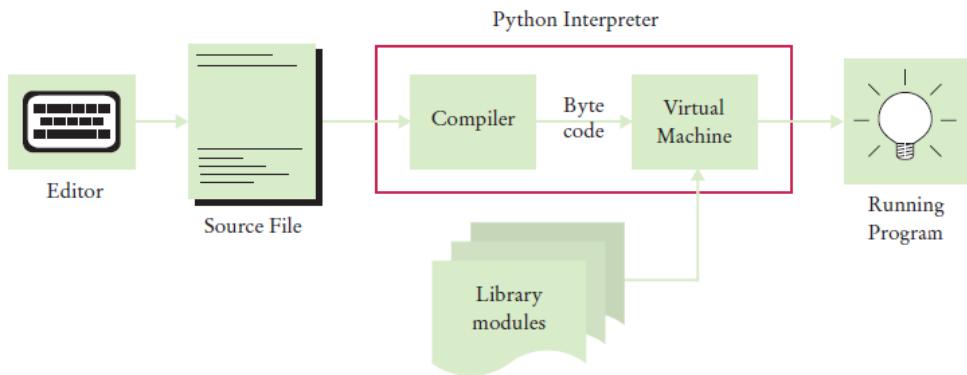
- คอมไพล์เวอร์ (Compiler) เป็นรูปแบบการแปลภาษาที่ต้องแปลโปรแกรมต้นฉบับ (Source code/Source program) ทั้งโปรแกรมให้เป็นภาษาเครื่องก่อนการทำงาน เช่น โดยระหว่างขั้นตอนการแปลภาษานี้ คอมไпал์เวอร์จะทำการตรวจสอบไวยากรณ์

ของภาษา (Syntax) ถ้ามีข้อผิดพลาดทางไวยากรณ์ของภาษาเกิดขึ้น (Syntax error) ก็จะแจ้งให้ทราบ เพื่อให้ผู้เขียนโปรแกรมแก้ไขให้ถูกต้องแล้วจึงแปลคำสั่งใหม่ เมื่อผ่านการแปลเรียบร้อยแล้วและไม่มีข้อผิดพลาดใดๆ ก็ตาม เรียกโปรแกรมส่วนที่แปลเสร็จเรียบร้อยในขั้นตอนนี้ว่า օบเจกต์โปรแกรม (Object program หรือ Object module) แต่ถ้าอุปกรณ์ที่ใช้ทำงานได้ทันที จำเป็นต้องผ่านกระบวนการลิงค์ (Link) หรือการรวมอุปกรณ์ที่สามารถทำงานได้ หรือเป็นภาษาเครื่องที่เรียกว่า เอ็กซ์เซกทูต์ (Execute program) หรือ โหลดโมดูล (Load module) ซึ่งโดยทั่วไปแล้วจะเป็นไฟล์ที่มีนามสกุลเป็น .exe หรือ .com และสามารถนำโปรแกรมนี้ไปใช้งานได้ตลอดโดยไม่ต้องแปลใหม่อีก และทำงานได้เร็ว แต่ถ้ามีการแก้ไขโปรแกรมต้นฉบับแม้เพียงเล็กน้อย ก็จำเป็นต้องทำการแปลใหม่ตั้งแต่ต้น ดังรูปที่ 1.2



รูปที่ 1.2 แสดงขั้นตอนการคอมไพล์โปรแกรม

- อินเทอร์พรีเตอร์ (Interpreter) เป็นรูปแบบการแปลภาษาในลักษณะบรรทัดต่อบรรทัด ในขณะโปรแกรมกำลังทำงาน (อินเทอร์พรีเตอร์จะทำการแปล และประมวลผลทีละคำสั่ง) ซึ่งแตกต่างจากการแปลภาษาแบบคอมไพล์ร์ ถ้าในระหว่างการแปลเกิดพบข้อผิดพลาดที่บรรทัดใดก็จะแสดงแจ้งเตือนให้ผู้เขียนโปรแกรมทราบทันที เพื่อให้ทำการแก้ไขให้ถูกต้องก่อนแปลบรรทัดถัดไป ดังนั้นมีจุดเด่นที่เรียกว่า “การทำงานช้ากว่าแบบคอมไابل์” แต่ประโยชน์ของภาษาที่ถูกแปลด้วยอินเทอร์พรีเตอร์ คือ โปรแกรมจะมีโครงสร้างที่ง่ายต่อการพัฒนา ไม่ซับซ้อน และอ่านทำความเข้าใจได้容易กว่า ตัวอย่างของภาษาที่มีการใช้อินเทอร์พรีเตอร์ ได้แก่ ภาษาพีเอชพี (PHP) ภาษาเพิร์ล (Perl) รวมถึงภาษาไพธอน (Python) ด้วย ดังแสดงในภาพที่ 1.3 จากสาเหตุที่รูปแบบของการแปลภาษาที่มีข้อเด่นข้อด้อยต่างกัน ดังนั้นจึงมีบางภาษาประยุกต์เอาข้อดีของ การแปลภาษาทั้งสองเข้าไว้ด้วยกัน อย่างเช่น ภาษาเบสิก เป็นต้น



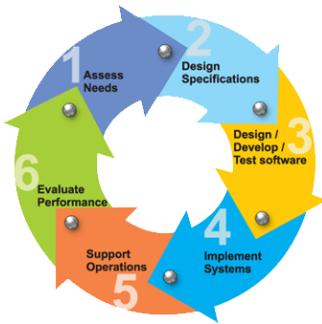
รูปที่ 1.3 แสดงขั้นตอนการทำงานของอินเตอร์พรีเตอร์

4. ขั้นตอนการพัฒนาโปรแกรม (Software development)

ในการพัฒนาโปรแกรมมีขั้นตอนที่เรียกว่า System development life cycle ดังนี้



- 1) ค้นหาความต้องการของระบบ (System requirements) คือ การศึกษา และเก็บความต้องการของผู้ใช้โปรแกรม ว่ามีความต้องการอะไรบ้าง
- 2) วิเคราะห์ระบบ (Analysis) คือ การนำเอาความต้องการของผู้ใช้โปรแกรมมาวิเคราะห์ว่าจะพัฒนาเป็นโปรแกรมตามที่ผู้ใช้ต้องการได้หรือไม่ ถ้าทำได้จะทำได้อย่างไรบ้าง
- 3) การออกแบบระบบ (Design) เมื่อสรุปได้แล้วว่าโปรแกรมที่จะสร้างมีลักษณะใด ขั้นตอนต่อมาคือ การออกแบบการทำงานของโปรแกรมให้เป็นไปตามความต้องการที่วิเคราะห์ไว้ การออกแบบอาจจะออกแบบเป็นผังงาน หรือรหัสเทียมก็ได้
- 4) การพัฒนาโปรแกรม (Coding) เมื่อได้ผังงานแล้ว ขั้นตอนต่อมาคือการเขียนโปรแกรมตามผังงานที่ได้ออกแบบไว้
- 5) การทดสอบ (System testing) เมื่อเขียนโปรแกรมเสร็จเรียบร้อยแล้ว จะต้องมีการทดสอบเพื่อหาข้อผิดพลาดต่างๆ ของโปรแกรม เช่น โปรแกรมตรวจความต้องการของผู้ใช้ หรือไม่ ถ้าพบข้อผิดพลาดก็กลับไปทำการออกแบบใหม่อีกรัง (กลับไปที่ขั้นตอนที่ 3 แต่ถ้าการวิเคราะห์ระบบไม่รัดกุมเพียงพอ บางครั้งอาจจะเป็นต้องย้อนกลับไปดำเนินการในขั้นตอนที่ 1 ใหม่ก็ได้)
- 6) การดูแลระบบ (Maintenance) เมื่อโปรแกรมผ่านการทดสอบแล้ว และผู้ใช้ได้นำโปรแกรมดังกล่าวไปใช้ ผู้พัฒนาจะต้องดูแล และแก้ไขข้อผิดพลาดที่หาไม่พบในขั้นตอนการทดสอบโปรแกรม สำหรับ System development life cycle แสดงในรูปที่ 1.4



รูปที่ 1.4 System development life cycle

5. หลักการเขียนผังงาน (Flowchart)



ผังงาน หรือ **Flowchart** เป็นการอธิบายขั้นตอนวิธีการดำเนินงานของระบบออกแบบอย่างเป็นระบบ (อยู่ในระหว่างขั้นตอนการวิเคราะห์ และการออกแบบระบบ) ซึ่งถูกเขียนโดยใช้รูปสัญลักษณ์ต่างๆ อธิบายถึง

กิจกรรมที่เกิดขึ้นในระบบงาน มีสันเชื่อมโยงระหว่างสัญลักษณ์อธิบายถึงความสัมพันธ์ และหัวลูกครอบคลุมทิศทางขั้นตอนการทำงาน เป็นต้น การอธิบายขั้นตอนการทำงานของระบบด้วยวิธีนี้ เป็นที่นิยมมากกว่าแบบอื่นๆ เนื่องจากสามารถอธิบายลำดับขั้นตอนการทำงานได้ชัดเจนมากกว่าวิธีการอื่นๆ เพราะผังงานส่วนใหญ่จะถูกแสดงด้วยภาพ สัญลักษณ์ สันเชื่อม ความสัมพันธ์ และกำกับด้วยข้อความที่สั้นกระชับ สามารถตรวจสอบความถูกต้องของระบบงานได้ง่าย ผังงานนิยมใช้เขียนแทนขั้นตอน คำอธิบาย ข้อความ หรือคำพูด ที่ใช้ในอัลกอริทึม (Algorithm) เนื่องจากมีจุดเด่นคือ สามารถอธิบายขั้นตอนการดำเนินงานระหว่างผู้ที่เกี่ยวข้อง ให้เข้าใจได้ตรงกันได้ดีกว่าการอธิบายด้วยคำพูด ข้อความ หรือเทคนิคการอธิบายแบบอื่นๆ ผังงานมี 2 ชนิด คือ

- 1) ผังงานระบบ (System flowchart) คือ ผังงานที่แสดงขั้นตอนการทำงานในระบบอย่างกว้างๆ แต่ไม่เจาะลึกในระบบงานย่อย
- 2) ผังงานโปรแกรม (Program flowchart) คือ ผังงานที่แสดงถึงขั้นตอนในการทำงานของโปรแกรม ตั้งแต่รับข้อมูล คำนวณ จนถึงแสดงผลลัพธ์

ประโยชน์ของผังงาน

1. สามารถทำความเข้าใจ และแยกแยะปัญหาในระบบงานได้ง่าย
2. แสดงลำดับการทำงานออกแบบมาให้เห็นได้อย่างชัดเจน
3. สามารถทำความเข้าใจระบบงานได้ง่าย และรวดเร็ว เมื่อเวลาผ่านไปนาน

4. ช่วยในการตรวจสอบ และแก้ไขโปรแกรมได้ง่าย เมื่อเกิดข้อผิดพลาด
5. ช่วยให้ผู้อื่นสามารถศึกษาการทำงานของโปรแกรมได้อย่างง่าย และรวดเร็วมากขึ้น
6. ไม่ขึ้นกับภาษาใดภาษาหนึ่ง

วิธีการเขียนผังงาน

- ใช้สัญลักษณ์ตามที่กำหนดไว้
- ใช้ลูกศรแสดงทิศทางการไหลของข้อมูลจากบันลังล่าง หรือจากซ้ายไปขวา ยกเว้นมีการทำงานแบบย้อนกลับ
- คำอธิบายในภาพควรมีข้อความที่สั้น กะทัดรัด และเข้าใจง่าย
- ทุกแผนภาพต้องมีลูกศรแสดงทิศทางเข้า - ออก
- ไม่ควรโยงเส้นเชื่อมผังงานที่อยู่ใกล้มาก ๆ ควรใช้สัญลักษณ์จุดเชื่อมต่อแทน
- ผังงานควรมีการทดสอบความถูกต้องของการทำงานก่อนนำไปเขียนโปรแกรม

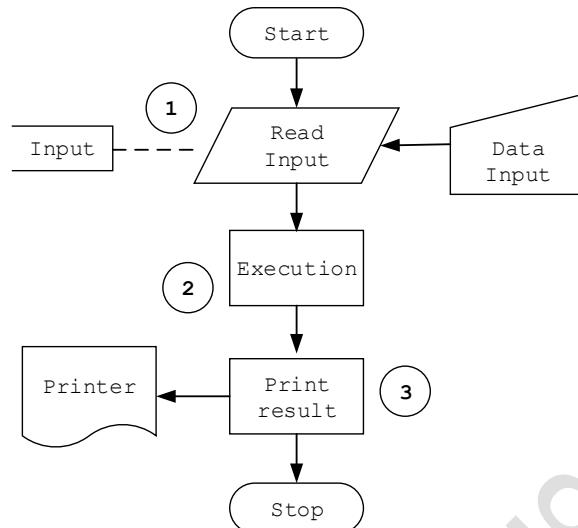
ตารางที่ 1.6 แสดงสัญลักษณ์ต่าง ๆ ที่ถูกใช้ในการเขียน Flowchart

ลำดับ	สัญลักษณ์	ชื่อเรียก	ความหมายของสัญลักษณ์
1		Start/End	แทนจุดเริ่มต้น และสิ้นสุดการทำงาน
2		Process	การดำเนินงานอย่าง ๆ ที่เกิดขึ้นในระบบงาน เช่น การประมวลผล การคำนวณ
3		Decision	เงื่อนไขการตัดสินใจ หรือทางเลือกในการดำเนินงาน
4		General	นำเข้าหรือส่งออกข้อมูล จากระบบ
5		Input/Output	คอมพิวเตอร์โดยมีระบุชนิดของอุปกรณ์
5		Subprocess/External Subroutine	ระบบย่อย พังชันย่อย หรือเรียกโปรแกรมจากภายนอก
6		Database	ฐานข้อมูลที่สามารถเก็บข้อมูลได้ในปริมาณมาก ใช้สำหรับรับข้อมูลเข้าระบบ หรือจัดเก็บ
7		External data	ข้อมูลอื่นที่นำเข้า-ส่งออก จากแหล่งข้อมูลอื่นภายนอก เช่น ข้อมูลจากต่างองค์กร ข้อมูลจากโปรแกรมอื่น ๆ
8		Manual input	รับข้อมูลจากผู้ใช้ที่กำหนดข้อมูลขึ้นมาใช้งานเอง (การนำข้อมูลเข้าด้วยมือ)

9		Display	แสดงผลออกทางจอภาพ
10		Document	ข้อมูลเป็นเอกสาร หรือแสดงข้อมูลออกทางเครื่องพิมพ์
11		Off-page reference	ต้องการเชื่อมโยงไปยังหน้าอื่นๆ กรณีที่ไม่สามารถเขียนผังงานได้ในหน้าเดียว
12		On-page reference	ต้องการเชื่อมโยงผังงานอื่นๆ แต่อยู่ในหน้าเดียวกัน
13		Arrow	เส้นทางการทำงาน ซึ่งจะปฏิบัติตามหัวลูกศรชี้
14		Manual operation	การทำงานด้วยมือ (ทำงานด้วยแรงคน)
15		Preparation	กำหนดชื่อตัวแปร ชนิดข้อมูล หรือการกำหนดค่าเริ่มต้นต่างๆ ก่อนการทำงาน
16		Sort	จัดเรียงข้อมูลตามข้อกำหนดที่ตั้งไว้
17		Comment	คำอธิบายการทำงาน

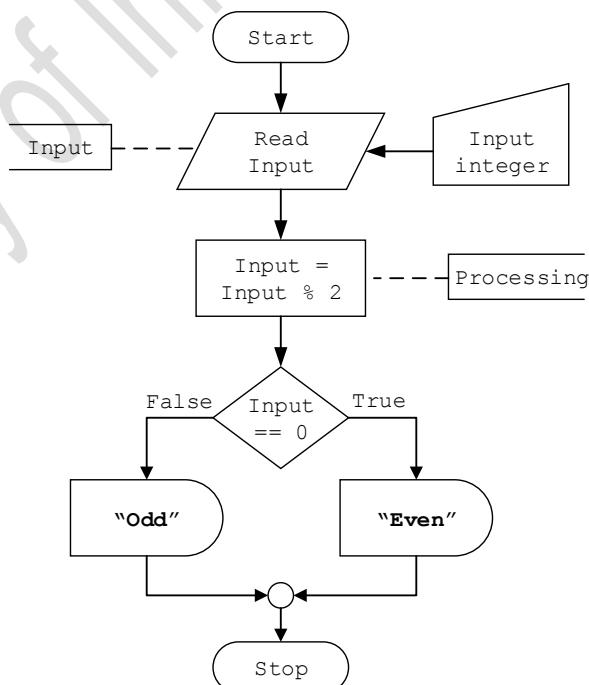
หลักการเขียนผังงาน ผู้อ่านจำเป็นต้องเข้าใจโครงสร้างการเขียน 3 แบบคือ โครงสร้างการทำงานแบบตามลำดับ โครงสร้างการทำงานแบบเลือกกระทำตามเงื่อนไข และโครงสร้างการทำงานแบบการทำซ้ำ ซึ่งมีหลักการเขียนในแต่ละประเภทดังต่อไปนี้

- โครงสร้างการทำงานแบบตามลำดับ (Sequence) เป็นรูปแบบการเขียน Flowchart ที่ง่ายที่สุด คือ เขียนให้การทำงานเริ่มต้นจากบนลงล่าง (การทำงานจะต่อเนื่องกันโดยเริ่มจากด้านบนลงมาอย่างด้านล่าง) เขียนคำสั่งทีละขั้นตอน หรือคำสั่งละ 1 บรรทัด โดยมีลูกศรชี้ลงจากด้านบนลงด้านล่างเพื่อแสดงลำดับการทำงาน เช่น สมมติให้งานชนิดหนึ่งมีกระบวนการทำงานเป็น 3 ขั้นตอน โดยขั้นตอนแรกคือ อ่านข้อมูลจากผู้ใช้ ขั้นตอนที่สองคือ การคำนวณ และขั้นตอนสุดท้ายคือ การพิมพ์ข้อมูลออกทางเครื่องพิมพ์ ซึ่งสามารถแสดงได้ดังรูปที่ 1.5



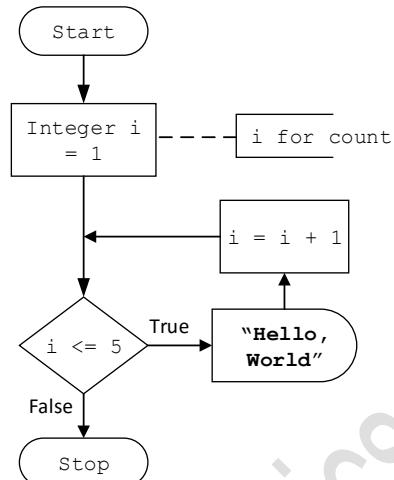
รูปที่ 1.5 โครงสร้างการทำงานแบบตามลำดับ

2. โครงสร้างการทำงานแบบเลือกตัดสินใจ หรือเงื่อนไข (Decision หรือ Selection) คือ การเขียน Flowchart เมื่อระบบงานเกิดทางเลือกที่จะต้องตัดสินใจ หรือเลือกกระทำไปในทิศทางใดทิศทางหนึ่ง โดยปกติจะมีเหตุการณ์ให้กระทำ 2 แบบคือ เงื่อนไขเป็นจริงจะกระทำการบวนการหนึ่ง หรือเป็นเท็จจะกระทำการอีกบวนการหนึ่ง แต่ถ้าการดำเนินงานมีความซับซ้อนมากขึ้น จะต้องใช้เงื่อนไขหลายชั้น เช่น การตัดเกรดนักศึกษา เป็นต้น จากตัวอย่างรูปที่ 1.6 แสดงผลการตัดสินใจอย่างง่าย คือ ต้องการตรวจสอบว่าเลขจำนวนเต็มที่รับเข้ามาเป็นเลขคี่หรือคู่ จากนั้นพิมพ์ข้อความว่าเป็นเลขจำนวนคู่หรือคี่



รูปที่ 1.6 โครงสร้างการทำงานแบบเลือกตัดสินใจ

3. โครงสร้างการทำงานแบบการทำซ้ำ (Repetition หรือ Loop) หมายถึงกระบวนการทำงานที่จำเป็นต้องทำแบบเดิมซ้ำๆ หลายๆ รอบ จนกว่าจะได้คำตอบที่ต้องการ หรือจนกว่าจะจบเงื่อนไขที่กำหนดไว้ ตัวอย่างเช่น ต้องการพิมพ์ข้อความ “Hello, World” ซ้ำกัน 5 ครั้ง ดังรูปที่ 1.7



รูปที่ 1.7 แสดงโครงสร้างการทำงานแบบการทำซ้ำ

ตัวอย่างการวิเคราะห์ระบบ และการเขียน Flowchart

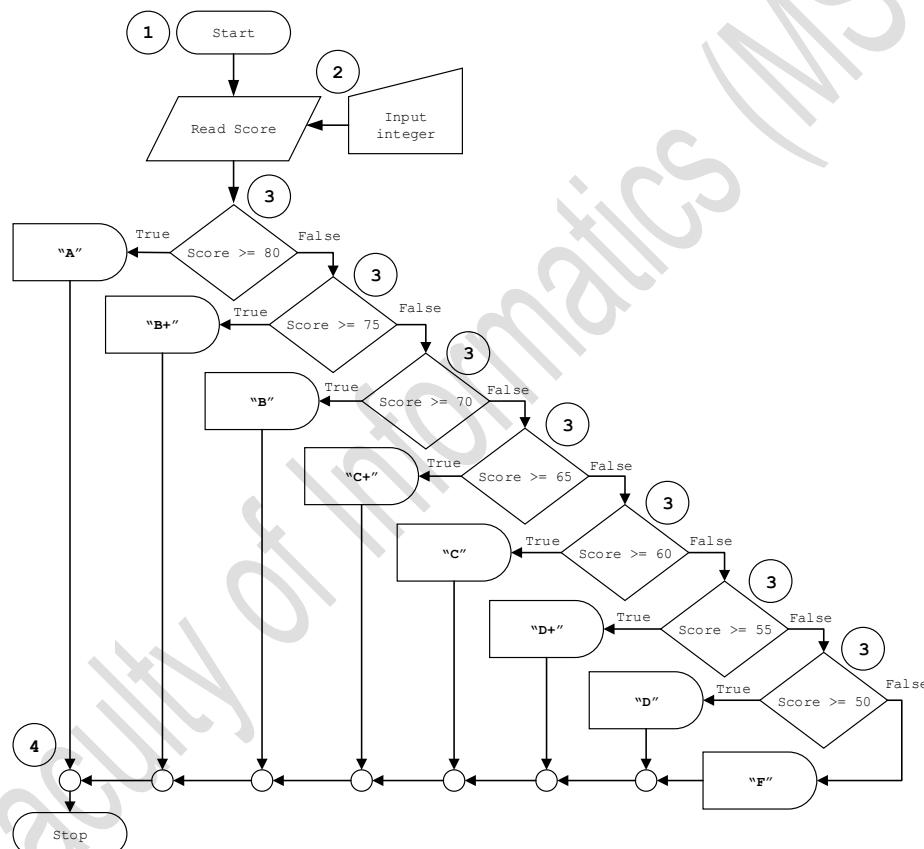
จงเขียน Flowchart เพื่อแสดงผลการเรียนเป็นเกรดแก่นักศึกษา โดยรับคะแนนจากแบบพิมพ์ และมีเงื่อนไขการให้เกรดดังเกณฑ์การให้คะแนนดังต่อไปนี้

คะแนน	เกรดที่ได้
80-100	A
75-79	B+
70-74	B
65-69	C+
60-64	C
55-59	D+
50-54	D
0-49	F

เริ่มต้นการเขียน Flowchart

1. เขียนสัญลักษณ์เริ่มต้น (ใช้สัญลักษณ์ลำดับที่ 1 ในตารางที่ 1.6) ดูภาพประกอบในรูปที่ 1.8

2. อ่านข้อมูลจากผู้ใช้งาน แล้วเก็บไว้ในตัวแปรชื่อ Score (ใช้สัญลักษณ์หมายเลขอ 4, 8 และ 13)
3. เปรียบเทียบว่าค่าข้อมูลที่อยู่ในตัวแปรชื่อ Score มีค่าอยู่ระหว่าง 80 – 100 หรือไม่ ถ้าใช่ ให้พิมพ์เกรด “A” ถ้าไม่ใช่ให้เปรียบเทียบกับค่าในตารางเกรด ไปเรื่อยๆ ถ้า ค่าใน Score อยู่ในช่วงของเกรดใดๆ ให้ทำการพิมพ์เกรดที่ได้ออกทางจอภาพ แต่ ถ้าไม่ใช่ หรือไม่เป็นจริง ให้เลื่อนมาเปรียบเทียบไปเรื่อยๆ จนกว่าจะไม่สามารถ เปรียบเทียบได้อีกจะพิมพ์เกรด F (ใช้สัญลักษณ์หมายเลขอ 3, 9 และ 13)
4. เมื่อไม่สามารถเปรียบเทียบท่อไปได้อีกแล้ว ให้จบการทำงาน (ใช้สัญลักษณ์ หมายเลขอ 1, 12 และ 13)



รูปที่ 1.8 แสดง Flowchart การคำนวณเกรดของนักศึกษา

6. หลักการเขียนรหัสเทียม หรือชูโดโค้ด (Pseudo Code)

รหัสเทียม (Pseudo code) คือ การเขียนขั้นตอนการดำเนินการของโปรแกรมให้อยู่ใน รูปแบบคำสั่งข้อความภาษาอังกฤษที่มีรูปแบบง่ายทั่วไป ไม่มีกฎเกณฑ์ที่แนนอนตามตัว

มองดูคล้ายภาษาอะไรดับสูงที่ใช้กับเครื่องคอมพิวเตอร์ ซึ่งไม่เจาะจงภาษาใดภาษานั่น รูปแบบคำสั่งที่นิยมใช้เป็นมาตรฐาน แสดงในตารางที่ 1.7

ตารางที่ 1.7 คำสั่งมาตรฐานสำหรับใช้เขียนรหัสเทียม

คำสั่ง (Keyword)	ความหมาย
WHILE	คำสั่งให้ทำงานซ้ำ โดยตรวจสอบเงื่อนไขก่อนเข้าทำงาน
IF-THEN-ELSE	คำสั่งให้เลือกทำอย่างใดอย่างหนึ่ง (จริงทำหลัง IF เท็จทำหลัง THEN)
REPEAT-UNTIL	คำสั่งให้ทำงานซ้ำ โดยอนุญาตให้ทำงานก่อนการตรวจสอบเงื่อนไข
CASE	คำสั่งให้เลือกตัดสินใจได้มากกว่า 2 ทางเลือก
FOR, LOOP	คำสั่งให้วางชั้วนกกว่าเงื่อนไขจะเป็นเท็จ
READ	คำสั่งอ่านข้อมูลเข้ามาเพื่อประมวลผล
COMPUTE	คำสั่งประมวลผลข้อมูล
SET, INIT	คำสั่งกำหนดค่าเริ่มต้นให้ตัวแปร
PRINT, DISPLAY	คำสั่งแสดงผลลัพธ์ของการประมวลผล
INCREASE	คำสั่งเพิ่มค่าให้กับตัวแปร
END	คำสั่งหยุดการทำงาน
ADD, SUB, +, -, *	คำสั่งดำเนินการทำคณิตศาสตร์

เกณฑ์ในการเขียนรหัสเทียม

1. ประโยชน์คำสั่งแต่ละคำสั่งให้เขียนเป็นภาษาอังกฤษที่ง่าย (ใช้ศัพท์ง่ายๆ)
2. เขียนประโยชน์คำสั่ง 1 คำสั่ง ต่อหนึ่งบรรทัดเท่านั้น
3. คำสั่งหลัก (Keyword) ควรเขียนตัวหนาเพื่อเน้นว่าต้องการทำอะไร
4. ควรเขียนคำสั่งให้มีลักษณะเป็นแบบย่อหน้า เพื่อแยกโครงสร้างการควบคุม และทำให้ผู้อ่านเข้าใจได้ง่าย
5. คำสั่งถูกเขียนจากบนลงล่างโดยมีทางเข้า-ออก เพียงทางเดียว

ขั้นตอนการเขียนรหัสเทียม

ลำดับขั้นตอนการเขียนรหัสเทียมเพื่อสั่งให้ระบบคอมพิวเตอร์ดำเนินการแบ่งออกเป็น 6 ขั้นตอนดังนี้ (ในการทำงานจริงอาจไม่จำเป็นต้องเรียงลำดับจาก 1 – 6 เสมอไป)

ขั้นตอนที่ 1: การกำหนดค่าให้กับตัวเก็บข้อมูล (ตัวแปร)

- การกำหนดค่าเริ่มต้นให้ตัวแปร ถือว่าเป็นขั้นตอนแรกสุดที่สมควรจะทำเป็นอย่างยิ่ง เพื่อเตรียมตัวก่อนการประมวลผล คำสั่งที่นิยมใช้ คือ INITIALIZE หรือ SET เช่น **SET Count = 1**
- การกำหนดค่าที่เกิดจากการประมวลผลไว้ในตัวแปรจะใช้เครื่องหมาย “=” เช่น **Count = Count + 1** หรือ **Sum = 50, Average = Sum / Count** เป็นต้น

ขั้นตอนที่ 2: การรับข้อมูล ทั้งจากคีย์บอร์ด ตัวแปร หรือจากแฟ้มข้อมูล คำสั่งที่นิยมใช้ คือ READ หรือ GET เช่น **READ Input**

ขั้นตอนที่ 3: การคำนวณทางคณิตศาสตร์ สามารถใช้เครื่องหมายทางคณิตศาสตร์ที่ใช้กันโดยทั่วไปได้ตามปกติ คือ +, -, *, /, %, ==, <>, () เป็นต้น เช่น $X = (G + 15) * 8/5$

ขั้นตอนที่ 4: การเปรียบเทียบ และการตัดสินใจ คำสั่งที่ใช้ คือ IF, THEN, ELSE

ขั้นตอนที่ 5: การดำเนินงานแบบทำซ้ำ คำสั่งที่ใช้ คือ WHILE, FOR, LOOP เป็นต้น

ขั้นตอนที่ 6: การแสดงผลลัพธ์ ทั้งจากจอภาพ เครื่องพิมพ์ อุปกรณ์ภายนอกอื่นๆ คำสั่งที่ใช้ คือ PRINT, WRITE, PUT, DISPLAY, OUTPUT เช่น **PRINT "Hello, Python"**

ตัวอย่างการเขียนรหัสเทียม สำหรับการตั้งมาใหม่ และการหาค่าเฉลี่ยของเกรดนิสิตจำนวน N คน

ตัวอย่างรหัสเทียมการตั้งมาใหม่ (ภาษาไทย)

1. เตรียมมาใหม่ไว้ 1 ช่อง
2. นิ่งซองมาใหม่ และเทลงถ้อยเปล่า
3. นิ่งซองเครื่องปั่น แล้วเทลงถ้อยเดิม
4. ตั้มนำร้อนให้เดือด 3 - 5 นาที แล้วเทลงถ้อย
5. ปิดฝา แล้วรอ 3 นาที
6. เปิดฝา แล้วรับประทาน

ตัวอย่างรหัสเทียมการหาค่าเฉลี่ยของเกรดนิสิตจำนวน N คน

1. **SET total to zero**
2. **SET grade counter to one**
3. **WHILE grade counter is less than or equal to N**
4. **INPUT the next grade**

5. **ADD** the grade into the total
6. **INCREASE** grade counter
7. **SET** the class average to the total divided by N
8. **PRINT** the class average.

ในตัวอย่างรหัสเที่ยมการหาค่าเกรดเฉลี่ยของนิสิต บรรทัดที่ 1 เป็นการกำหนดค่า total ให้มีค่าเป็น 0 โดยตัวแปรดังกล่าวใช้เพื่อเก็บค่าผลรวมของเกรดจำนวน N ตัว บรรทัดที่ 2 กำหนดค่าให้ grade counter มีค่าเป็น 1 ใช้เพื่อเป็นตัวนับจำนวน บรรทัดที่ 3 เป็นการตรวจสอบเงื่อนไขว่า grade counter มีค่าน้อยกว่า หรือเท่ากับ N จริงหรือไม่ ถ้าเป็นจริง จะทำในบรรทัดที่ 4 ต่อไป แต่ถ้าเป็นเท็จ โปรแกรมจะกระโดดไปทำงานบรรทัดที่ 7 ซึ่งบรรทัดนี้ เป็นการกำหนดค่าให้ตัวแปร class average ซึ่งค่าที่กำหนดให้ตัวแปรดังกล่าวเป็นผลมาจากการนำเอาค่าที่เก็บในตัวแปร total หารด้วย N (สมการการหาค่าเฉลี่ย) และบรรทัดที่ 8 เป็นการพิมพ์ผลลัพธ์ค่าเฉลี่ยออกทางเตอร์พุต สำหรับในบรรทัดที่ 4 เป็นการรับค่าเกรดของนิสิตจากอินพุต และบรรทัดที่ 5 จะเป็นคำสั่งรวมเกรดของนิสิตแต่ละคนเข้าไว้ด้วยกัน และเก็บลงในตัวแปร total และบรรทัดที่ 6 เป็นการเพิ่มค่าให้ตัวแปร grade counter ขึ้นครั้งละ 1 จนกว่าเงื่อนไขการวนรับข้อมูลเกรดจะเป็นเท็จ นั่นคือจำนวนนิสิตมากกว่า N นั่นเอง

จบทบที่ 1

บทที่ 2

โปรแกรมภาษาไพธอน (Python programming language)



1. ภาษาไพธอนคืออะไร (What's python language)

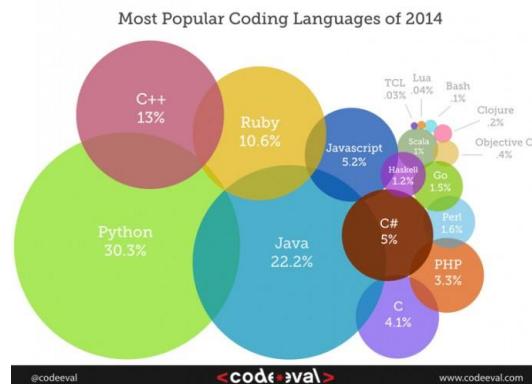


“ไพธอน (Python = งูเหลือม ตามความหมายในพจนานุกรม) คือ ภาษาระดับสูงที่ใช้ในการพัฒนาโปรแกรมอีกภาษาหนึ่งที่มีความสามารถสูงไม่แพ้ภาษาอื่นๆ ที่มีอยู่ในปัจจุบัน ถูกสร้างขึ้นโดยนักพัฒนาโปรแกรมชื่อ Guido van Rossum เป็นชาวดัชท์ (Dutch) ประเทศเนเธอร์แลนด์ เกิดเมื่อวันที่ 31 มกราคม พ.ศ. 2499 ภาษาไพธอนได้รับอิทธิพลมาจากการภาษา ABC ซึ่งมีความสามารถในการจัดการเกี่ยวกับข้อผิดพลาดของโปรแกรม (Exception handling) ได้ดี และดึงเอาความสามารถเด่นๆ ของภาษาระดับสูงอื่นๆ มาประยุกต์ดัดแปลงใช้กับไฟร่อนด้วย ส่งผลให้ภาษาไพธอนเป็นที่นิยม และใช้งานกันอย่างกว้างขวางในปัจจุบัน เนื่องจากเป็นภาษาที่สามารถเรียนรู้ได้ง่าย รวดเร็ว รูปแบบการเขียนโปรแกรมมีความกระหัตต์ และมีประสิทธิภาพสูง จากการนำเสนอคุณลักษณะเด่นๆ ของภาษาอื่นๆ มาเป็นพื้นฐานในการพัฒนาต่ออยอดนี้เอง ไพธอนจึงถูกเรียกว่าเป็นภาษาที่มีหลายกระบวนการทัศน์ หรือหลายมุมมอง (Multi-paradigm languages) ซึ่งเป็นการผสมผสานรวมเอาแนวทางความคิดในการพัฒนาซอฟต์แวร์แบบต่างๆ เข้าไว้ด้วยกันให้อยู่ในตัวของไพธอน คือ Object-oriented programming, Structured programming, Functional programming และ Aspect-oriented programming

2. คุณสมบัติและความสามารถของภาษาไพธอน (Features and capability of Python)

ไพธอนถูกพัฒนาขึ้นมาโดยไม่ขึ้นกับแพลตฟอร์ม (Platform independent) กล่าวคือ สามารถทำงานได้ทั้งบนระบบปฏิบัติการตระกูลวินโดว์ (Windows NT, 2000, 2008, XP, 7, 8, 8.1) ตระกูลยูนิกซ์-ลินก์ (Unix, Linux, xBSD) และตระกูลแมคด้วย (Macintosh) โดยระบบปฏิบัติการเหล่านี้ติดตั้งเพียงโปรแกรมแปลงภาษาให้เป็นภาษาเครื่องของสถาปัตยกรรมนั้นๆ เท่านั้น ดูข้อมูลเพิ่มเติมเกี่ยวกับแพลตฟอร์มที่ไพธอนสนับสนุนได้ที่เว็บของไพธอน <http://www.python.org>

ภาษาไพธอนเป็นซอฟต์แวร์เสรี (Open source software) เหมือนภาษาพีเอชพี (PHP) ทำให้ทุกคนสามารถนำไฟร่อนมาพัฒนาโปรแกรมได้ฟรีๆ โดยไม่ต้องเสียค่าใช้จ่าย และคุณสมบัติความเป็นซอฟต์แวร์เสรี ทำให้มีโปรแกรมหรือว่าโลกเข้ามาร่วมกันพัฒนาให้ไฟร่อนมีความสามารถสูงขึ้นเรื่อยๆ ส่งผลให้สามารถครอบคลุมงานในลักษณะต่างๆ อย่างกว้างขวาง สามารถสรุปคุณสมบัติ และความสามารถของภาษาไฟร่อนได้ดังต่อไปนี้



คุณสมบัติเด่นของภาษาไฟร่อน

- โปรแกรมตันฉบับที่ถูกเขียนขึ้นด้วยภาษาไฟร่อน สามารถนำไปประมวลผลได้กับหลายระบบปฏิบัติการ (Portable/Cross platform/Platform independent) เช่น Unix, Linux, Microsoft-windows (NT, 2000, 2003, 2008, 2012, 95, 98, ME, XP, 7, 8, 8.1), Amiga, Macintosh, BeOS, AIX, AROS, AS/400, OS/2, xBSD, VMS, QNX, MS-DOS, OS/390, z/OS, Palm OS, PlayStation, Psion, Solaris, RISC OS, HP-UX, Pocket PC และ VMS เป็นต้น
- ตัวภาษาไฟร่อนถูกสร้างขึ้นมาจากภาษาซี ทำให้ได้รับอิทธิพลไวยกรณ์ทางภาษาติดมากจากภาษาซีด้วย ดังนั้นผู้ที่คุ้นเคยกับการเขียนโปรแกรมภาษาซีสามารถปรับตัวในเขียนภาษาไฟร่อนได้ไม่ยาก
- ภาษาไฟร่อนเป็นภาษาที่สวยงาม ง่ายต่อการเรียนรู้ (Readability) เขียนโปรแกรมได้กระชับ (Writability) เนื่องจากมีโครงสร้างของภาษาไม่ซับซ้อนเข้าใจง่าย เป็นภาษาที่มีความยืดหยุ่นสูงมาก (Flexibility) และมีความเสถียรภาพ (Reliability)
- ไฟร่อนมีความสามารถในการจัดการหน่วยความจำอัตโนมัติ (Garbage collection) สามารถบริหารจัดการพื้นที่หน่วยความจำที่ใช้งานแบบไม่ต่อเนื่องให้สามารถทำงานได้อย่างมีประสิทธิภาพ ทำให้ผู้เขียนโปรแกรมไม่ต้องกังวลเกี่ยวกับการคืนหน่วยความจำคืนให้กับระบบเหมือนภาษาซี

- การแปลภาษาของภาษา Python เป็นแบบอินเทอร์พรีเตอร์ เป็นภาษาสคริปต์ คือจะประมวลผลไปทีละบรรทัด ทำให้ใช้เวลาในการเขียนโปรแกรม และการคอมไพล์ไม่มาก หมายความว่ากับงานด้านการดูแลระบบ (System administration) เป็นอย่างยิ่ง
- ไวยากรณ์อ่านง่าย เนื่องจากภาษา Python ได้กำหนดการใช้สัญลักษณ์ที่ใช้ในการกำหนดขอบเขต {...} ของโปรแกรมออกໄປ (สำหรับผู้ที่เขียนภาษา C หรือ Java มา ก่อน ในตอนแรกๆ จะไม่ค่อยชอบนัก แต่เมื่อเขียนไปเรื่อยๆ จะรู้สึกว่าคล่องตัวกว่า) โดยใช้การย่อหน้าแทน ทำให้สามารถอ่านโปรแกรมที่เขียนได้ง่าย นอกจากนั้นยังมีการสนับสนุนการเขียน docstring ซึ่งเป็นข้อความสั้นๆ ที่ใช้อธิบายการทำงานของฟังก์ชัน คลาส และโมดูล ได้ด้วย
- Python เป็นภาษาglue language คือสามารถเรียกใช้ภาษาอื่นๆ ได้หลายภาษา ทำให้เหมาะสมที่จะใช้เขียนเพื่อประสานงานกับโปรแกรมที่เขียนในภาษาอื่นๆ ได้ดี
- ภาษา Python ถูกสร้างขึ้นโดยรวมคุณสมบัติเด่นๆ ของภาษาต่างๆ เข้ามาไว้ด้วยกัน อาทิเช่น ภาษา C, C++, Java, Perl, ABC, Modula-3, Icon, Matlab, ANSI C, Lisp, Smalltalk และ Tcl เป็นต้น
- Python สามารถเรียกใช้ภาษา C/C++ ได้ ในการกลับกันภาษา C/C++ ก็อนุญาตให้ผังชุดคำสั่งของ Python เอาไว้ภายในภาษา C/C++ ได้ เช่นเดียวกัน
- ไม่ต้องเสียค่าใช้จ่ายใดๆ ทั้งสิ้น เพราะตัวแปรภาษา Python อยู่ภายใต้ลิขสิทธิ์ GNU หรือซอฟต์แวร์เสรี
- ภาษา Python และชุดของไลบรารี สนับสนุนการประมวลผลทางด้านวิทยาศาสตร์ และวิศวกรรมศาสตร์ ได้อย่างมีประสิทธิภาพ
- มีความยืดหยุ่นสูง ทำให้การจัดการกับงานด้านข้อมูล และ Text File ได้เป็นอย่างดี
- Python มีฟังก์ชันที่สนับสนุนการเชื่อมต่อกับระบบฐานข้อมูล ได้หลากหลายชนิด เช่น MySQL, Sybase, Oracle, Informix, ODBC และอื่นๆ
- Python สนับสนุนการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)
- Python นำเสนอด้วยโครงสร้างตัวแปรใหม่ (Built-in Object Types) เพื่อให้ผู้เขียนโปรแกรมสะดวกในการพัฒนางานมากขึ้น เช่น ลิสต์ (List) ดิกชันnaire (Dictionary) ทัพเบิล (Tuple) และเซ็ต (Set) เป็นต้น ซึ่งโครงสร้างตัวแปรใหม่เหล่านี้ ทำให้ง่ายต่อการใช้งาน และมีประสิทธิภาพสูง
- Python เตรียมเครื่องมือต่างๆ เพื่อใช้ในการประมวลผลข้อมูลประเภทเท็กซ์ไฟล์ การจัดเรียงข้อมูล การเชื่อมต่อสตริง การตรวจสอบเงื่อนไขของข้อมูล การแทนค่า ไว้อย่างครบถ้วน

- ภาษา Python เป็นภาษาประเภท Server side script คือ การทำงานของภาษา Python จะทำงานด้านผู้เซิร์ฟเวอร์ (Server) และส่งผลลัพธ์กลับมายังไคล์ล์เอนท์ (Client) ทำให้มีความปลอดภัยสูง
- Python มีโมดูลสนับสนุนการเขียนโปรแกรมกับระบบ (System) เช่น โพรเซส เครด รวมถึงระบบเครือข่ายคอมพิวเตอร์ ได้เป็นอย่างดี
- Python เตรียมเครื่องมือสำหรับสร้าง Internet script หรือ CGI script สำหรับเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตผ่านซ็อกเก็ต (Sockets API) จึงทำให้สามารถเชื่อมต่อ และใช้งานแอพพิเคชันต่างๆ แบบระยะไกลได้ เช่น FTP, Gopher, SSH เป็นต้น
- สนับสนุนเทคโนโลยี Component Object Model Technologies (COM) ของวินโดว์ส CORBA, ORBs, XML เป็นต้น
- Python จัดเตรียมเครื่องมือสำหรับจัดการงานด้าน Regular Expression (กลุ่มของสัญลักษณ์ที่ใช้ในการค้นหา แทนที่ หรือเปลี่ยนเทียบคำ/ข้อความกับข้อมูลชนิดสตริง) ไว้อย่างเพียงพอ
- ภาษา Python ใช้พัฒนาเว็บเซอร์วิส (Web service) รวมทั้งใช้บริหารการสร้างเว็บไซต์ สำเร็จรูปที่เรียกว่า Content Management Framework (CMF)
- Python อนุญาตให้ผู้พัฒนาโปรแกรมสามารถสร้าง Dynamic Link Library (DLL) จากภาษาอื่นๆ เพื่อใช้งานร่วมกับ Python ได้ เช่น .dll ของวินโดว์ส เป็นต้น
- Python ใช้มาตราฐานสำหรับสร้างอนเตอร์เฟส คือ Tkinter API ที่ได้รับอิทธิพลมาจาก Tcl/Tk ที่ทำงานบนยูนิกส์มาก่อน ซึ่งสนับสนุนกราฟฟิกของ X windows, วินโดว์ส และ Macintosh จุดเด่นที่สำคัญของการใช้ Tkinter API คือช่วยให้ผู้พัฒนาโปรแกรมไม่จำเป็นต้องแก้ไขรหัสต้นฉบับเมื่อนำไปทำงานบนระบบปฏิบัติการอื่นๆ
- Python มีไลบรารีที่สนับสนุนงานด้านการสร้างภาพกราฟฟิก และการประมวลผลภาพ (Image processing) มากมาย เช่น การปรับความคมชัดของภาพ การอ่านไฟล์ภาพขนาดใหญ่ การบันทึกไฟล์ในรูปแบบต่างๆ ได้อย่างสะดวก และมีประสิทธิภาพ
- Python เตรียมไลบรารีสำหรับสนับสนุนการเขียนโปรแกรมทางด้านปัญญาประดิษฐ์ (Artificial intelligent) เช่น Naive Bayes และ K-Nearest Neighbors ไว้ด้วย
- Python สนับสนุนการทำงานแบบ Dynamic typing คือ สามารถเปลี่ยนชนิดของข้อมูลได้ง่าย และสะดวก
- สนับสนุนให้สามารถเขียนโปรแกรมจากร่วมกับ Python ได้ โดยใช้ Jython โดยทำงานอยู่บน Java Virtual Machine (JVM) ได้

- สนับสนุนการเขียนโปรแกรมร่วมกับ .NET Framework ของไมโครซอฟต์ โดยการใช้ IronPython
- ไฟร่อนสนับสนุนการสร้างเอกสาร PDF โดยไม่ต้องติดตั้ง Acrobat Writer
- ไฟร่อนสนับสนุนการสร้าง Shockwaves Flash (SWF) โดยไม่จำเป็นต้องติดตั้ง Macromedia Flash

คุณสมบัติด้อยของภาษาไฟร่อน

- **ความเร็ว:** ไฟร่อนเป็นภาษาสคริปต์ (Scripting language) ซึ่งทำงานโดยมีตัวแปลงภาษา (Interpreter) แปลงคำสั่งในแต่ละบรรทัดของโปรแกรมต้นฉบับ (Source code) ให้เป็นภาษาเครื่อง (Machine code) ในขณะที่โปรแกรมกำลังทำงาน ซึ่งแตกต่างจากภาษาซี C++ โอบอล หรือปาสคาล เพราะภาษาเหล่านี้จะทำการแปลรหัสต้นฉบับให้ลายเป็นภาษาเครื่องทั้งหมดก่อนเริ่มต้นทำงาน ส่งผลให้โปรแกรมขนาดใหญ่ที่เขียนขึ้นด้วยภาษาไฟร่อนจะทำงานได้ช้ากว่าโปรแกรมที่ใช้เทคนิคการคอมไพล์แฟ้มต้นฉบับทั้งหมดก่อน
- **โอกาสเกิดข้อผิดพลาดชนิด Runtime Error สูงขึ้น:** จุดด้อยในข้อนี้ มีผลกระทบมาจากการแปลงภาษาแบบ Interpreter และการไม่ได้แปลรหัสต้นฉบับทั้งหมดก่อนทำงานนั้นเอง ในการประมวลผลของภาษาสคริปต์ จะไม่มีการตรวจสอบความถูกต้องของการเรียกใช้ตัวแปร และชนิดของตัวแปรทั้งหมดก่อนเริ่มทำงาน ดังนั้น ถ้าผู้พัฒนาโปรแกรมขาดความระมัดระวัง (Logic error) ในระหว่างพัฒนาโปรแกรม จะทำให้มีโอกาสเกิดความผิดพลาด จากการเรียกใช้ตัวแปรที่ไม่ได้ประกาศไว้ หรือใช้งานตัวแปรผิดประเภทได้ง่าย
- **การระบุขอบเขตของคำสั่ง และตัวแปร:** ไฟร่อนไม่ใช้ [...] เป็นสัญลักษณ์สำหรับกำหนดขอบเขตของคำสั่ง หรือตัวแปรในการเขียนโปรแกรม เมื่อเทียบกับภาษา如 C/C++ และ Java แต่ใช้การย่อหน้าเพื่อบอกขอบเขตของคำสั่ง และตัวแปรแทน ส่งผลให้ยากต่อการพัฒนาโปรแกรมที่มีขนาดใหญ่ และโปรแกรมต้นฉบับมีความซับซ้อนมาก ๆ เช่น nested loop เพราะต้องสังเกตการย่อหน้าให้ถูกต้อง



Note: แม้ว่าโปรแกรมที่ถูกพัฒนาด้วยไฟร่อนจะมีความเร็วในการทำงานช้ากว่า C/C++ และ Java ก็ตาม แต่ไฟร่อนมีข้อเด่นที่สำคัญ คือ ความเร็วในการพัฒนาโปรแกรม ปัจจุบันมีโปรแกรมเมอร์พยายามจะนำข้อดีของไฟร่อน คือ ความยืดหยุ่น และความง่ายในการเขียน โปรแกรม รวมกับความเร็วในการทำงานของ C/C++ เข้าไว้ด้วยกัน เช่น cython, Psyco, py2exe, cx_Freeze, py2app, bbfreeze, IronPython เป็นต้น

3. ประวัติการสร้างภาษาไฟร่อน (History of Python)

สำหรับประวัติการพัฒนาภาษา Python นั้นต้องย้อนกลับไปช่วงปลายปี 1980 (พ.ศ. 2523) ซึ่งเป็นช่วงที่มีความคิดริเริ่มที่จะสร้างภาษา Python แต่การเริ่มต้นการพัฒนาจริงๆ เกิดขึ้นในช่วงเดือนธันวาคมปี 1989 โดยนักพัฒนาโปรแกรมชื่อ Guido van Rossum ทำงานที่ CWI (National Research Institute for Mathematics and Computer Science) เป็นชาวดัชท์ ประเทศเนเธอร์แลนด์ โดย Python ได้รับอิทธิพล และความสำเร็จมาจากการโปรแกรมภาษา ABC ที่ณ ในเวลานั้นมีความสามารถเกี่ยวกับการจัดการข้อผิดพลาดของโปรแกรม (exception handling) และการเชื่อมต่อ (Interfacing) กับระบบปฏิบัติการ Amoeba operating system ได้เป็นอย่างดี โดย Guido van Rossum ถือว่าเป็นผู้ริเริ่มและคิดค้น แต่เขามีความคิดว่าผลงานอย่าง Python ที่เขารังสรรค์ เป็นผลิตผลที่เกิดขึ้นจากความสนุกสนานเป็นหลัก โดยเขาได้ยังงานชิ้นนี้ว่าเป็น Benevolent Dictator for Life (BDFL = ผลงานที่ถูกสร้างขึ้นจากความสนุกสนาน) เพราะเป็นผลงานชิ้นแรกจะเกิดจากความไม่ได้ตั้งใจ และอยากทำงานที่เป็นอิสระนั้นเอง ซึ่งคนที่ถูกกล่าวถึงว่าทำงานในลักษณะแบบนี้มาก่อนก็ได้แก่ Linus Torvalds ผู้สร้าง Linux kernel, Larry Wall ผู้สร้าง Perl programming language นั้นเอง

กุมภาพันธ์ ค.ศ. 1991 (พ.ศ. 2534) Van Rossum ได้เผยแพร่ภาษา Python ออกมารังแรกคือ เวอร์ชัน 0.9.0 ขณะนั้นเขากำลังพัฒนา Python ในส่วนของ การสืบทอดคุณสมบัติ (inheritance) การจัดการความผิดพลาด (exception handling) พังชัน ชนิดของตัวแปรหลักๆ เช่น list, dict, str และตัวแปรอื่นๆ สำหรับใช้กับ Python ในอนาคต และเปิดเผยแพร่ส่วนที่ใช้บริหารจัดโมดูลของระบบ (Module system) โดยยึดคุณสมบัติที่ได้มาจากการ Modula-3

มกราคม ค.ศ. 1994 (พ.ศ. 2537) ต่อมาอีกประมาณ 3 ปี Van Rossum จึงได้เผยแพร่ Python ในเวอร์ชันที่ 1.0 ออกมานี้มีคุณสมบัติหลักๆ ที่เพิ่มเติมขึ้นมาคือ lambda, reduce, filter และ map ซึ่งบางส่วนได้นำเอาคุณสมบัติมาจากภาษา Lisp มาประยุกต์ใช้งาน

ค.ศ. 1995 (พ.ศ. 2538) Python เวอร์ชัน 1.2 นั้นได้ถูกปล่อยออกมายโดยขณะนั้น Van Rossum ยังคงพัฒนา Python อยู่ที่ CWI เนเธอร์แลนด์ ต่อมาเขาได้ย้ายไปพัฒนา Python ต่อที่ Corporation for National Research Initiatives (CNRI) เรสตัน มลรัฐเวอร์จิเนีย ประเทศสหรัฐอเมริกา ขณะที่เขางานอยู่ที่ CNRI เขายังได้พัฒนา และปล่อย Python เวอร์ชันต่างๆ ออกมากหลายรุ่น ดังนี้คือ 1.4, 1.5 และรุ่น 1.6 ซึ่งเพิ่มคุณสมบัติของ Python ให้สูงขึ้น เช่น เพิ่มความสามารถในการจัดการกับจำนวนเชิงซ้อน (Complex number), data hiding และ name mangling เป็นต้น ซึ่งหลังจากปล่อยรุ่น 1.6 ออกมานั้น Guido van Rossum ก็ได้ออกจาก CNRI เพื่อทำธุรกิจพัฒนาซอฟต์แวร์แบบเต็มตัว โดยก่อนที่จะเริ่มทำงานธุรกิจ เขายังได้ทำให้ Python นั้นมีลิขสิทธิ์แบบ General Public License (GPL) โดยที่ CNRI และ Free Software Foundation (FSF) ได้ร่วมกันเปิดเผยรหัสโปรแกรมทั้งหมด เพื่อให้ Python นั้นได้ชื่อว่า เป็นซอฟต์แวร์เสรี และเพื่อให้ตรงตามข้อกำหนดของ GPL-compatible ด้วย (แต่ยังคงไม่สมบูรณ์ เพราะ

การพัฒนาในรุ่น 1.6 นั้นออกแบบก่อนที่จะใช้สัญญาลิขสิทธิ์แบบ GPL ทำให้ยังมีบางส่วนที่บังเปิดเผยแพร่ไม่ได้ และในปีเดียวกันนั้นเอง Guido van Rossum ก็ได้รับรางวัลจาก FSF ในชื่อว่า "Advancement of Free Software" โดยในปีนั้นเอง Python 1.6.1 ก็ได้ออกมาเพื่อแก้ปัญหาข้อผิดพลาดของตัวซอฟต์แวร์ และให้เป็นไปตามข้อกำหนด ของ GPL-compatible license อย่างสมบูรณ์

คุณภาพ ค.ศ. 2000 (พ.ศ. 2543) ในปีนี้ Van Rossum และ Python Core Development team ได้ย้ายไปทำงานร่วมกับ BeOpen.com และได้เผยแพร่ Python เวอร์ชันที่ 2.0 สู่สาธารณะ บนเว็บไซต์ของ BeOpen.com หลังจากนั้น Van Rossum และนักพัฒนา Python คนอื่นๆ ที่อยู่ในทีมที่ชื่อว่า PythonLabs ก็ได้จับมือร่วมกันทำงานในภายใต้ชื่อทีมว่า Digital Creations ซึ่งเวอร์ชันนี้ได้เพิ่มความสามารถของ Python ให้สนับสนุน List ที่ได้ลอกเลียนแบบมาจากภาษา SETL และการจัดการหน่วยความจำคืนให้ระบบ (Garbage collection)

เมษายน ค.ศ. 2001 (พ.ศ. 2544) คลอด Python เวอร์ชัน 2.1 ซึ่งได้สืบทอด และพัฒนามาจาก 1.6.1 เกือบทั้งหมด และจาก Python เวอร์ชัน 2.0 บางส่วน Python ได้ทำการเปลี่ยนชื่อสัญญาลิขสิทธิ์ใหม่เป็น Python Software Foundation License โดยที่ใน Python เวอร์ชัน 2.1 รหัส alpha เป็นสิทธิ์ของ Python Software Foundation (PSF) ซึ่งเป็นองค์กรที่ไม่หวังผลกำไร เช่นเดียวกับ Apache Software Foundation เวอร์ชันนี้ได้เพิ่มคุณสมบัติของ Nested scopes

ธันวาคม ค.ศ. 2001 เวอร์ชัน 2.2 ปรับโครงสร้างการเขียนโปรแกรมคล้ายภาษา C

กรกฎาคม ค.ศ. 2003 เวอร์ชัน 2.3

พฤษภาคม ค.ศ. 2004 เวอร์ชัน 2.4

กันยายน ค.ศ. 2006 เวอร์ชัน 2.5

สิงหาคม ค.ศ. 2008 เวอร์ชัน 2.6

กันยายน ค.ศ. 2010 เวอร์ชัน 2.7

ธันวาคม ค.ศ. 2008 เวอร์ชัน 3.0 หรือเรียกว่า Python 3000 หรือ Py3K ถูกออกแบบเพื่อแก้ไขข้อบกพร่องบางอย่างในการออกแบบขั้นพื้นฐานของภาษา เช่น การจัดคุณลักษณะที่ซ้ำซ้อนของเวอร์ชันก่อนหน้า ปัจจุบันภาษา Python เผยแพร่อยู่ร่องรอย คือเวอร์ชัน 3.4.0 beta2 (มกราคม 2014) โดยเพิ่มเติมความสามารถใหม่ๆ เพิ่มขึ้นเรื่อยๆ สามารถอ่านข้อมูลเพิ่มเติมได้ที่ <http://www.python.org>

4. งานด้านต่างๆ ที่ Python สนับสนุนในปัจจุบัน (Application domains)

ตามที่ได้กล่าวไว้แล้วในตอนต้นว่าภาษา Python เป็นภาษาแบบผสมผสานหลายระบบหัวหน้า (Multi-paradigm language) ดังนั้นจึงรองรับการพัฒนาได้หลากหลายประเภท ดังนี้

1. สนับสนุนการพัฒนางานเกี่ยวกับเว็บแอปพลิเคชัน (Web application) สำหรับงานในส่วนของซอฟต์แวร์ขนาดใหญ่ ใช้พัฒนา Common Gateway Interface (CGI), เพرمเวิร์คของ Django และ TurboGears, โซลูชันระดับสูงอย่างเช่น Zope และ Mega รวมไปถึง Content Management Systems ขั้นสูงอย่าง Plone และ CPS เป็นต้น ส่วนผู้ของไซต์เฉพาะ (Custom web solutions) ไฟร่อนก็สนับสนุนงานในหลายๆ ด้าน เช่น HTML และ XML, E-mail, RSS feeds และไฟร์โทกอลเกี่ยวกับงานอินเทอร์เน็ตอื่นๆ
2. สนับสนุนงานด้านฐานข้อมูล (Database Access) ไฟร่อนสามารถเชื่อมต่อกับฐานข้อมูลได้มากมายหลายค่าย โดยผ่านทาง ODBC Interfaces และผ่านทาง Standard database API ของไฟร่อนโดยตรง เช่น MySQL, Oracle, MS SQL Server, PostgreSQL, SybODBC, ZODB, Durus และอื่นๆ ที่จะมีเพิ่มเติมขึ้นอีกในอนาคต
3. สนับสนุนงานด้าน Desktop GUIs ตัวติดตั้งของไฟร่อนมี Tk GUI development library อยู่ในตัวอยู่แล้ว ดังนั้นผู้เขียนโปรแกรมสามารถสร้างกราฟฟิกได้ทันทีโดยไม่ต้องติดตั้งซอฟต์แวร์อื่นๆ เพิ่มเติม ซึ่งไลบรารีดังกล่าวเหมือนกับ Microsoft Foundation Classes, wxWidgets, GTK, Qt, pyside, Delphi และอื่นๆ ทำให้สามารถพัฒนาซอฟต์แวร์ประยุกต์ต่างๆ แบบกราฟฟิกได้ง่าย
4. สนับสนุนงานด้านวิทยาศาสตร์ และการประมวลผลตัวเลข (Scientific and Numeric computation) ไฟร่อนรองรับการทำงานด้านนักวิทยาศาสตร์ วิศวกรรมศาสตร์ เช่น ทฤษฎีการคำนวณ คณิตศาสตร์ เป็นต้น
5. สนับสนุนงานด้านชีวสารสนเทศศาสตร์ (Bioinformatics) เช่น Bioinformatics และ Physics เป็นต้น
6. สนับสนุนงานด้านการโปรแกรมเครือข่าย (Network programming) ไฟร่อนสามารถเขียนโปรแกรมในระดับต่ำเพื่อเชื่อมต่อกับระบบเครือข่ายได้เป็นอย่างดี ง่ายต่อการพัฒนาโดยอาศัย Sockets สามารถทำงานร่วมกับโมดูลอย่าง Twisted และ Framework สำหรับสร้าง Asynchronous network programming เป็นต้น
7. สนับสนุนด้านการศึกษา (Education) ไฟร่อนเหมาะสมสำหรับการเรียนการสอนในทุกสาขาวิชาที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์ สามารถนำหลักสูตรที่ Python Software Foundation เตรียมไว้ คือ pyBiblio และ Software Carpentry Course นำมาสอนได้ทันที หรือแทรกเข้าไปในรายวิชาดังกล่าวเพื่อเสริมความรู้ได้
8. สนับสนุนด้านการพัฒนาเกมส์ และงานด้าน 2, 3D (Game and 2/3D Graphics Rendering) ช่วงเวลาที่ผ่านมาไฟร่อนถูกใช้ในการพัฒนาเกมส์ทั้งเชิงธุรกิจ และสมัครเล่น โดยมีการสร้าง Framework สำหรับพัฒนา Game บนไฟร่อนโดยเฉพาะซึ่ง

เรียกว่า PyGame และ PyKyra นอกจากนี้ Python ยังมี libraries ที่มีความสามารถในการทำ 3D Graphics Rendering ออยู่มาก many

9. การพัฒนาซอฟต์แวร์ (Software development) ไฟรอนนั้นสนับสนุนการพัฒนาซอฟต์แวร์ที่มีการควบคุมคุณภาพ โดยมีเครื่องมือในการพัฒนาที่ดีมากับไฟรอนเอง เช่น Scons สำหรับสร้างรหัสเครื่อง (Build), Buildbot และ Apache Gump ที่ใช้สำหรับงาน Automated continuous compilation และ Testing, Roundup หรือ Trac สำหรับการค้นหาข้อผิดพลาด (bug tracking) และการบริหารจัดการโครงการไฟรอนซอฟต์แวร์ (Project management)

5. ตัวอย่างหน่วยงานที่เลือกใช้ไฟรอน แบ่งตามประเภทต่าง ๆ ได้ดังนี้

1. หน่วยงานเกี่ยวกับเว็บแอ�플ิเคชัน เช่น Yahoo Maps, Yahoo Groups, Google, Zope Corporation, Ultraseek, Linux Weekly News, ElasticHosts Cloud Servers, Shopzilla, Multiplayer.it เป็นต้น



2. หน่วยงานเกี่ยวกับการสร้างเกมส์ (Games) เช่น Battlefield 2, Crystal Space, Star Trek Bridge Commander, Civilization 4, QuArK (Quake Army Knife) เป็นต้น



3. หน่วยงานเกี่ยวกับงานกราฟฟิก (Graphics) เช่น Industrial Light & Magic, Walt Disney Feature Animation, HKS, Inc. (ABAQUS/CAE), Blender 3D เป็นต้น



4. หน่วยงานเกี่ยวกับสถาบันการเงิน (Financial) เช่น Altis Investment Management, ABN AMRO Bank, Treasury Systems, Bellco Credit Union เป็นต้น



5. หน่วยงานเกี่ยวกับงานด้านวิทยาศาสตร์ (Science) เช่น National Weather Service, Applied Maths, The National Research Council of Canada, NASA, Swedish Meteorological and Hydrological Institute (SMHI), Environmental Systems Research Institute (ESRI), Nmag Computational Micromagnetics, Objexx Engineering เป็นต้น



6. หน่วยงานเกี่ยวกับงานด้านการออกแบบอุปกรณ์อิเล็กทรอนิกส์ เช่น Ciranova, Productivity Design Tools เป็นต้น



7. หน่วยงานเกี่ยวกับงานด้านการพัฒนาซอฟต์แวร์ (Software development) เช่น Object Domain, Red Hat, SGI, Inc., MCI Worldcom, Nokia เป็นต้น



8. หน่วยงานเกี่ยวกับงานด้านการศึกษา (Education) เช่น University of California, Irvine, Delft University of Technology, Faculty of Aerospace Engineering, Delft, Netherlands, Smeal College of Business, The Pennsylvania State University, New Zealand Digital Library, IT Certification Exam preparation เป็นต้น



9. หน่วยงานเกี่ยวกับงานด้านซอฟต์แวร์ธุรกิจ เช่น Raven Bear Systems Corporation, Thawte Consulting, Advanced Management Solutions Inc., IBM, RealNetworks, dSPACE, Escom, Nexedi, Nektra, WuBook เป็นต้น
10. หน่วยงานของรัฐบาล เช่น the USA Central Intelligence Agency (CIA)



6. ตัวอย่างซอฟต์แวร์ที่เขียนด้วยไพธอน

1. บิตทอร์เรนต์ (BitTorrent)
2. Chandler โปรแกรมจัดการข้อมูลส่วนบุคคล
3. บางส่วนของ GNOME

4. บางส่วนของ Blender
5. Mailman โปรแกรมจัดการจดหมายกลุ่ม (เมลลิงลิสต์)
6. MoinMoin โปรแกรมวิกิ
7. Portage ส่วนจัดการแพกเกจของ Gentoo Linux
8. Zope แอปพลิเคชันเชิร์ฟเวอร์
9. เทอร์บอเกียร์ และ Django สำหรับพัฒนาโปรแกรมประยุกต์บนเว็บ และอีกมากมาย
สามารถอ่านเพิ่มเติมได้ที่ <https://wiki.python.org/moin/Applications>



7. การติดตั้งและใช้งานไฟรอนบนระบบปฏิบัติการวินโดว์

1. ดาวน์โหลดไฟล์ติดตั้งของไฟรอนได้จาก <http://www.python.org> ดังรูป 2.1

[Read more, -or- download Python now](#)

» **Python 3.4.0 beta 1 has been released**
The first beta for Python 3.4, [Python 3.4.0b1](#), has been released.
Published: Sun, 24 November 2013, 14:00 -0800

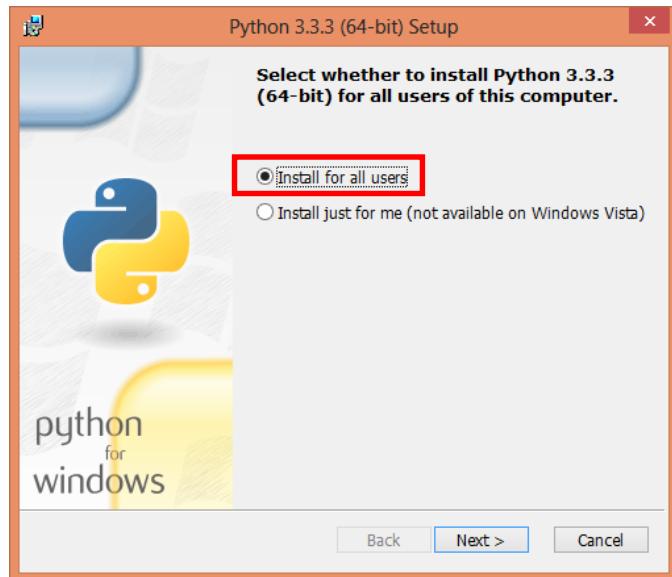
» **Python 3.3.3 released**
Python 3.3.3 is now available.
Published: Tue, 19 November 2013, 08:00 +0100

» **Python 3.3.3 release candidate 2 has been released**
The [second release candidate for Python 3.3.3](#) has been released.
Published: Tue, 12 November 2013, 08:00 +0100

รูปที่ 2.1 ดาวน์โหลดไฟรอนสำหรับวินโดว์

คลิกดาวน์โหลดที่ Python เวอร์ชันล่าสุด (ในที่นี้คือ Python 3.3.3 stable) ต่อจากนั้นให้คลิกเลือกที่ Windows X86-64 MSI Installer (3.3.3) เพื่อดาวน์โหลดตัวติดตั้งซึ่งจะมีส่วนขยายเป็น .msi ขนาดของไฟล์ประมาณ 20 เมگกะไบต์

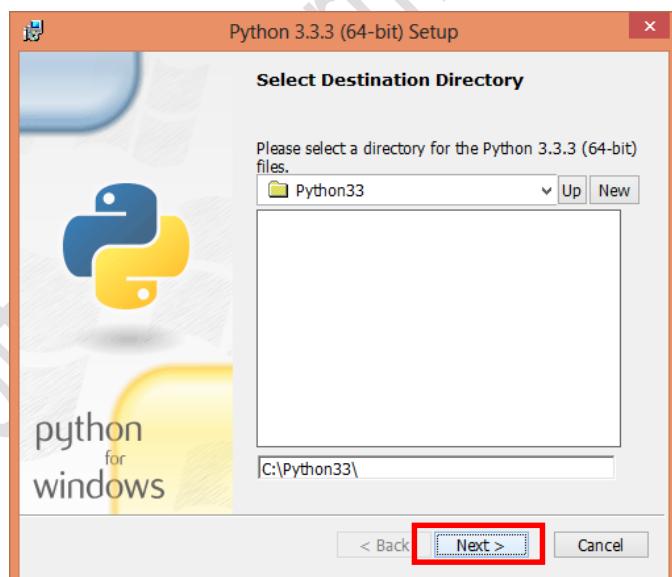
2. เมื่อได้ไฟล์ python-3.3.3.msi แล้ว ให้ดับเบิลคลิกเพื่อเข้าสู่ขั้นตอนการติดตั้ง ดังรูปที่ 2.2



รูปที่ 2.2 เริ่มต้นการติดตั้งไพธอน

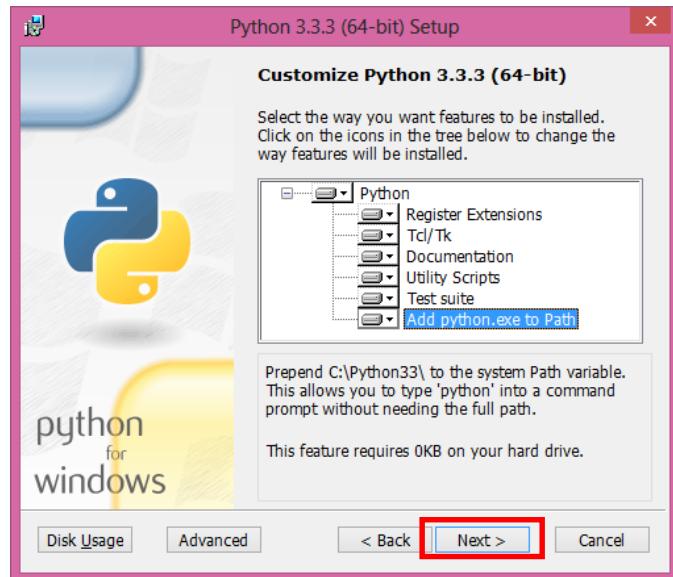
ให้เลือก Install for all users และคลิก Next >

3. จะเข้าสู่หน้าจอเลือก Path ในการติดตั้ง ดังรูปที่ 2.3 ให้เลือก C:\Python33 และคลิก Next >



รูปที่ 2.3 เลือกไดเรคทรอรีที่จะติดตั้งไพธอน

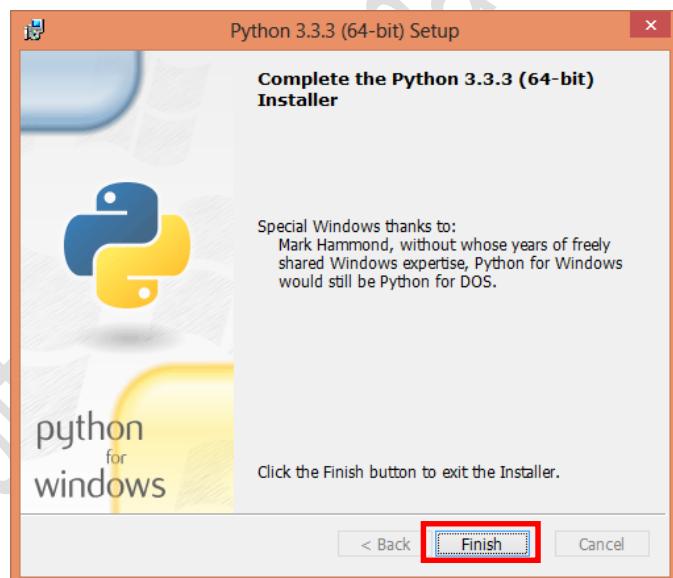
4. จะเข้าสู่หน้าจอ เลือก Features ในการติดตั้ง ให้เลือกติดตั้งตามค่า默認แล้วคลิก Next > ในรูป 2.4



รูปที่ 2.4 เลือก Features

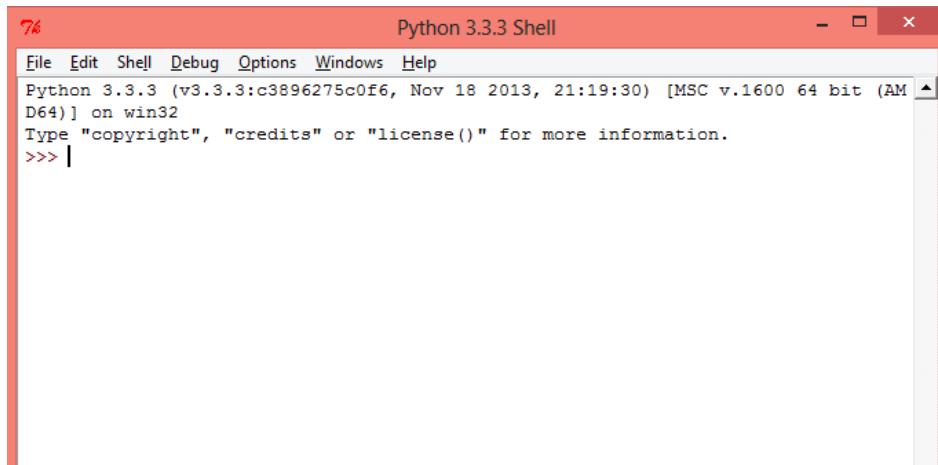
รอจนติดตั้งเสร็จ จะได้หน้าจอดังรูปที่ 2.5

- คลิก ปุ่ม Finish เป็นการสิ้นสุดขั้นตอนการติดตั้ง



รูปที่ 2.5 สิ้นสุดการติดตั้ง Python

- ทดสอบใช้งานโปรแกรมภาษา Python เข้าไปที่ Start >> Program >> Python3.3 >> IDLE (Python GUI) จะได้หน้าจอ ดังรูปที่ 2.6



รูปที่ 2.6 แสดงไฟรอนพร้อมใช้งาน (Prompt หรือ Python shell)

8. การติดตั้งและใช้งานไฟรอนระบบปฏิบัติการยูนิกซ์-ลินุกซ์

โดยปกติไฟรอนจะติดตั้งมากับลินุกซ์อยู่แล้ว ทดสอบโดยการเปิดเทอร์มินอล และพิมพ์คำสั่ง `python@root:~$ python` จะแสดงเวอร์ชันของไฟรอนออกมาก ถ้าไม่ปรากฏแสดงว่ายังไม่ได้ติดตั้งเอาไว้ วิธีการติดตั้งทำได้ 2 วิธีคือ ติดตั้งแบบ package และแบบ Manual

การติดตั้งแบบ package บนลินุกซ์ Centos/Fedora

1. พิมพ์คำสั่งต่อไปนี้ในเทอร์มินอล

```
python@root:~$ yum install openssl-devel bzip2-devel expat-devel gdbm-devel  
readline-devel sqlite-devel
```

รอสักครู่ โปรแกรมจะติดตั้ง package ต่างๆ ให้อัตโนมัติ เมื่อไม่มีข้อผิดพลาดใดๆ เกิดขึ้น ไฟรอนจะสามารถเรียกใช้งานได้ทันทีโดยใช้คำสั่ง `python@root:~$ python`



Note: การติดตั้งแบบ Package ต้องเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตเสมอ

การติดตั้งแบบ Package บนลินุกซ์ Ubuntu/ LinuxMint/Debian

1. พิมพ์คำสั่งต่อไปนี้ในเทอร์มินอล

```
python@root:~$ sudo apt-get install build-essential libncursesw5-dev libreadline5-dev libssl-dev libgdbm-dev libc6-dev libsqlite3-dev tk-dev
```

รอสักครู่ โปรแกรมจะติดตั้ง package ต่างๆ ให้อัตโนมัติ

การติดตั้งแบบ Manual บนลินุกซ์-ยูนิกซ์

การติดตั้งแบบ Manual เป็นการติดตั้งด้วยการนำเอาไฟล์ต้นฉบับ (Source program) มาคอมไพล์ กับระบบปฏิบัติการ บนสถาปัตยกรรมของเครื่องปัจจุบันที่กำลังทำงานอยู่ ดังนั้นจำเป็นต้องมีคอมไпал์เลอร์ของภาษา C/C++ ติดตั้งบนเครื่องเสียก่อน เพราะว่าไฟรอนถูกเขียนด้วยภาษา C นั่นเอง โดยปกติลินุกซ์จะติดตั้งคอมไพล์เลอร์ภาษา C มาอยู่แล้ว ทดสอบโดยใช้คำสั่งดังนี้ในเทอร์มินอล

```
python@root:~$ gcc หรือ g++
```

ขั้นตอนการติดตั้งไฟรอนแบบ Manual

- ดาวน์โหลดแฟ้มต้นฉบับของไฟรอนเวอร์ชันล่าสุดที่

<http://www.python.org/download/releases> เลือกเวอร์ชันที่ต้องการดาวน์โหลด
จากนั้นใช้คำสั่ง (เวอร์ชันล่าสุดในที่นี่คือ Python-3.3.3.tar.bz2)

```
python@root:~$ wget http://www.python.org/ftp/python/3.3.3/Python-3.3.3.tar.bz2
```

- แตกไฟล์ที่บีบอัดไว้ในไดเรคทรอรี่ปัจจุบัน (ทดสอบว่าอยู่ในไดเรคทรอรี่ใดๆ ด้วยคำสั่ง pwd)

```
python@root:~$ tar -xjf Python-3.3.3.tar.bz2
```

เมื่อแตกไฟล์บีบอัดแล้วจะมีไฟล์เดอร์ชื่อว่า Python-3.3.3 pragky ให้ทำการเปลี่ยนที่อยู่ไปในไฟล์เดอร์ดังกล่าว ด้วยคำสั่ง cd

```
python@root:~$ cd Python-3.3.3
```

- ถอนฟิกเส้นทางการติดตั้ง และคอมไпал์แฟ้มต้นฉบับด้วยคำสั่ง

```
python@root:~$ ./configure --prefix=/opt/python3
```

```
python@root:~$ make
```

```
python@root:~$ sudo make install
```

ถึงขั้นตอนนี้อาจจะต้องใช้เวลานานพอสมควรขึ้นอยู่กับความเร็วของคอมพิวเตอร์ เมื่อติดตั้งเสร็จโดยไม่มีข้อผิดพลาดใดๆ เกิดขึ้น ไฟรอนจะถูกติดตั้งอยู่ที่ไฟล์เดอร์ /opt/python3 ทดสอบเรียกใช้งานโดยใช้คำสั่งดังนี้

```
python@root:~$ /opt/python3/bin/python3 -V
```

ถ้าปรากฏข้อความแสดงเวอร์ชันของ Python แสดงว่าการติดตั้งสำเร็จแล้ว

9. การติดตั้งและใช้งาน Python บนระบบปฏิบัติการ macOS

โดยปกติ Python จะติดตั้งอยู่บนเครื่อง Mac มาแล้ว เช่นเดียวกัน ทดสอบโดยใช้เรียก Python ผ่าน เทอร์มินอลด้วยคำสั่ง [localhost:~] you% **python** (เทอร์มินอลของ Mac เรียกใช้งานโดยเลือกที่ /Applications/Utilities และดับเบิลคลิกที่ Terminal) แต่ถ้าไม่แสดงเวอร์ชัน แสดงว่ายังไม่มีการติดตั้งไว้ ซึ่งสามารถติดตั้งได้ตามลำดับขั้นตอนดังนี้

1. ดาวน์โหลด MacPython-OSX disk image จาก <http://homepages.cwi.nl/~jack/macpython/download.html>
2. คลิกเพื่อดาวน์โหลดไฟล์ชื่อ MacPython-OSX-2.3-2.dmg เก็บไว้บน desktop
3. ดับเบิลคลิกที่ไฟล์ดังกล่าวเพื่อทำการมาส์กอเมจ
4. ดับเบิลคลิกที่ไฟล์ชื่อ MacPython-OSX.pkg
5. การติดตั้งจะต้องใช้สิทธิ์ของ admin โดยจะมี prompt ขึ้นมาเพื่อให้ป้อนชื่อ และรหัสผ่าน
6. หลังจากติดตั้งแล้ว ให้เปิดโฟลเดอร์ /Application/MacPython-2.3
7. ดับเบิลคลิกที่ไฟล์ชื่อ PythonIDE จะปรากฏ prompt ของ Python เป็น >>> แสดงว่า พร้อมรับคำสั่งเรียบร้อยแล้ว

10. การใช้งาน Python บน Python shell (IDLE: Python GUI) และ Command line บนวินโดว์

ในหัวข้อนี้จะแนะนำถึงวิธีการเขียนโปรแกรม Python บนเซลล์ (เป็นเครื่องมือที่ใช้สำหรับเขียน โปรแกรมต้นฉบับ และรันโปรแกรม) ที่ติดตั้งมาพร้อมกับโปรแกรมภาษา Python ที่ได้ติดตั้งไปแล้วใน หัวขอก่อนหน้า (ในที่นี้เน้นการใช้งานเซลล์บนวินโดว์เท่านั้น) โดย Python ได้เตรียมเซลล์ให้ผู้ที่ต้องการ เขียน Python ไว้ 2 วิธี และอีกวิธีคือ ผู้ใช้สั่งรันโปรแกรมด้วยตัวเองผ่านทาง Command line

วิธีการใช้งาน Python shell (IDLE: Python GUI)

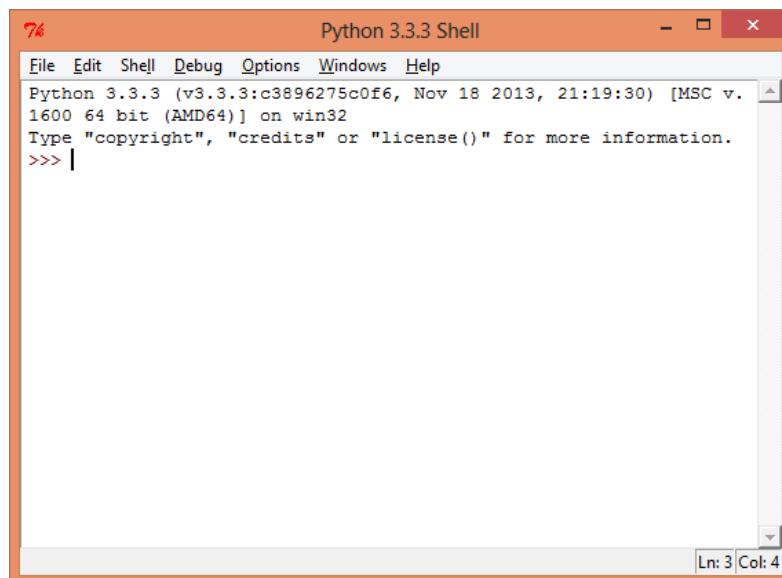
Python shell จะเป็นโปรแกรมที่ Python เตรียมไว้ให้สำหรับเขียนโปรแกรม และรันโปรแกรม มี ลักษณะการทำงานคล้าย Editor ของโปรแกรม Matlab คือ ผู้ใช้ป้อนคำสั่งทีละคำสั่ง เมื่อกดปุ่ม Enter

โปรแกรมจะประมวลผลทันที (ทำงานในลักษณะบรรทัดต่อบรรทัด) โดยมีสัญลักษณ์ของ prompt คือ

>>>

การเรียกใช้งานโปรแกรม Python shell สำหรับวินโดวัส 98, XP, 7 โดยเลือกที่ Start ⇒ Programs ⇒ Python 3.3 ⇒ IDLE (Python GUI)

สำหรับระบบปฏิบัติการเป็นวินโดวัส 8, 8.1 ให้กดปุ่ม  (อยู่ที่ด้านซ้ายล่างของคีย์บอร์ด) ⇒ พิมพ์คำว่า python ⇒ IDLE (Python GUI) จะปรากฏหน้าต่างโปรแกรม Python shell ดังรูปที่ 2.7



รูปที่ 2.7 Python shell

ผู้พัฒนาโปรแกรมสามารถป้อนคำสั่งได้ทันที ทดสอบการทำงานของชุดโดยป้อนคำสั่งต่อจาก สัญลักษณ์ >>> และกดปุ่ม Enter

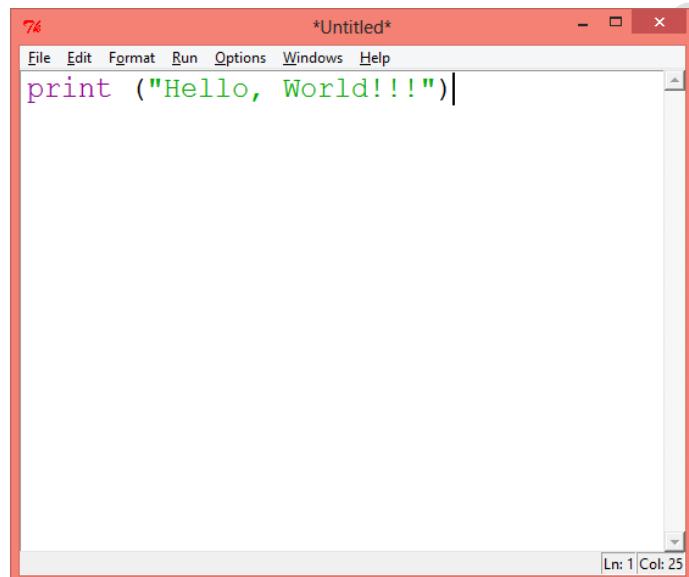
```

>>> print ("Hello, World!!!")      ↵
Hello, World!!!
>>> 2 + 5
7
>>> a = 10
>>> b = 5.0
>>> print(a+b)                 # a + b = 10 + 5.0 = 15.0
15.0
>>> str = "Python"             #String assignment
>>> str
'Python'
>>> print (str)
Python
>>> a <= b                   # Logic operation
False
>>> not (a and b or a)       # Expression

```

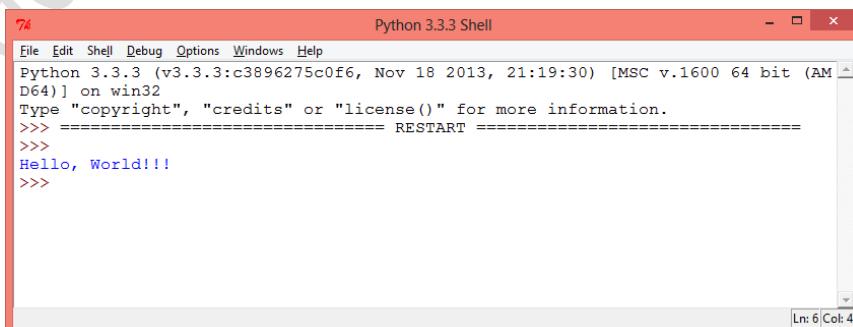
```
False
>>> a, b, c = 1.0, 3, .5      # Multiple assignment
>>> print (a,b,c)
1.0 3 0.5
```

ในการนี้ที่ผู้พัฒนาโปรแกรมต้องการเขียนโปรแกรมมากกว่า 1 คำสั่ง พร้อมกัน สามารถทำได้โดยเลือกที่เมนู File \Rightarrow New File หรือกดปุ่ม Ctrl + N ก็ได้ ไปสอนจะทำการเปิด Editor ขึ้นมาอีกด้วยหนึ่ง เพื่อให้ผู้พัฒนาสามารถเขียนโปรแกรมได้หลายๆ คำสั่งพร้อมกัน ซึ่ง Editor ดังกล่าวจะทำการบันทึกแฟ้มต้นฉบับมีนามสกุล .py ดังนั้นก่อนที่ผู้เขียนโปรแกรมจะส่งรันโปรแกรมจำเป็นต้องมีการบันทึกแฟ้มต้นฉบับเสียก่อน ดังรูปที่ 2.8



รูปที่ 2.8 การเขียนโปรแกรมบน Python text editor

สำหรับวิธีการรันโปรแกรมทำได้โดยคลิกเลือกที่เมนู Run \Rightarrow Run Module หรือกดปุ่ม F5 จากนั้นโปรแกรมจะแจ้งเตือนให้บันทึกแฟ้มต้นฉบับเสียก่อน ให้เลือกที่ OK \Rightarrow เลือกที่เก็บแฟ้มข้อมูล และตั้งชื่อโปรแกรม (ส่วนขยายจะเป็น .py อัตโนมัติ) \Rightarrow เลือก Save โปรแกรมจะแสดงผลดังรูปที่ 2.9



รูปที่ 2.9 แสดงการรันโปรแกรม

วิธีการใช้งาน Python ผ่านทางคอมมานด์ไลน์ (Command line)

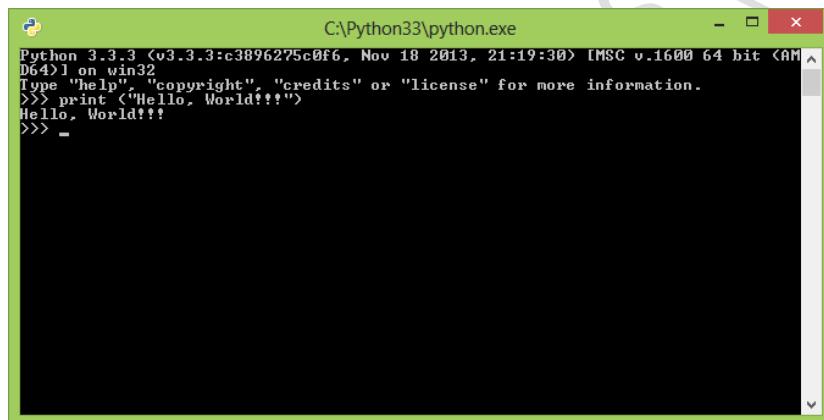
ไฟรอนได้จัดเตรียมเครื่องมือที่ใช้เขียนโปรแกรม และสั่งรันโปรแกรมผ่านทางระบบปฏิบัติการ ดอส (MS-DOS) สำหรับนักพัฒนาโปรแกรมที่ยังคุ้นเคยกับการใช้dosอยู่ ซึ่งสามารถเรียกใช้โปรแกรมได้ดังนี้

การเรียกใช้งานโปรแกรม Python (Command line) สำหรับวินโดวส์ 98, XP, 7 โดยเลือกที่

Start \Rightarrow Programs \Rightarrow Python 3.3 \Rightarrow Python (Command line)

เมื่อระบบปฏิบัติการเป็นวินโดวส์ 8, 8.1 ให้กดปุ่ม  (อยู่ที่ด้านซ้ายล่างของคีย์บอร์ด)

\Rightarrow พิมพ์คำว่า python \Rightarrow Python (Command line) จะปรากฏหน้าต่างโปรแกรม Python (Command line) ดังรูปที่ 2.10



รูปที่ 2.10 แสดงโปรแกรม Python (Command line)

สำหรับการใช้งาน Python (Command line) จะเหมือนกับการใช้ Python shell ทุกประการ

ปัญหาของ Python shell และ Python (Command line)

Python shell และ Command line นั้นไม่รองรับการส่งค่าตัวแปรชนิด argv ดังนั้น เมื่อผู้พัฒนาโปรแกรมต้องการที่จะส่งค่าตัวแปร ให้ทำงานพร้อมกับโปรแกรมขณะทำการสั่งรัน สามารถทำได้โดยการเรียกใช้งานผ่าน MS-DOS โดยตรงจากวินโดวส์ ดังนี้

สำหรับวินโดวส์ 98, XP, 7 เลือกที่ Start \Rightarrow Run \Rightarrow พิมพ์ cmd หรือ command

ระบบปฏิบัติการเป็นวินโดวส์ 8, 8.1 ให้กดปุ่ม  ⇒ พิมพ์ cmd หรือ command ดังรูปที่

2.11



รูปที่ 2.11 MS-DOS command

ทดสอบรันโปรแกรมด้วยคำสั่ง python บน MS-DOS command ถ้าได้ติดตั้ง Python ไว้แล้วจะแสดงข้อความแสดงเวอร์ชันให้เห็น แต่ถ้าไม่มีข้อความดังกล่าวออกมากแสดงว่า ยังไม่ได้ติดตั้ง Python หรือติดตั้งไว้แล้วแต่ไม่ได้ทำการบอกเส้นทาง (Set path) ไว้ให้กับระบบปฏิบัติการได้ทราบ ให้ผู้อ่านย้อนกลับไปอ่านในหัวข้อ การติดตั้ง และใช้งาน Python บนระบบปฏิบัติการวินโดวส์ อีกครั้งอย่างละเอียด

การสั่งรันโปรแกรมพร้อมกับส่งค่าตัวแปร argv, argc

โดยปกติ โปรแกรมเกื่อนทุกๆ ภาษาจะอนุญาตให้ผู้ใช้สามารถส่งค่าตัวแปรไปพร้อมกับการสั่งรันโปรแกรมได้ เรียกตัวแปรชนิดนี้ว่า อาคิวเมนต์ (Argument) ซึ่งมีอยู่ 2 ตัวคือ argc และ argv จากตัวอย่างโปรแกรมภาษา C++ ด้านล่าง แสดงการใช้งานตัวแปร argc และ argv

```
#include <iostream>

int main (int argc, char** argv) {
    std::cout << "Have " << argc << " arguments:" << std::endl;
    for (int i = 0; i < argc; ++i) {
        std::cout << argv[i] << std::endl;
    }
}
```

เมื่อสั่งรันโปรแกรมด้วยคำสั่ง ./Lab1 para1 para2 ซึ่งจะให้ผลลัพธ์ในการทำงานดังนี้

```
Have 3 arguments:
./Lab1
para1
para2
```

ตัวแปร argc จะทำการเก็บจำนวนของพารามิเตอร์ทั้งหมด เริ่มตั้งแต่ชื่อโปรแกรมไว้ ในที่นี่คือ 3 (Lab1 = ตัวที่ 1, para1 = ตัวที่ 2 และ para2 คือตัวที่ 3) ในขณะที่ตัวแปร argv เป็นชนิดอาร์เรย์ จะทำการเก็บพารามิตเตอร์ทุกตัวเริ่มตั้งแต่ชื่อโปรแกรมไว้ คือ argv[0] = Lab1, argv[1] = para1 และ argv[2] = para2

สำหรับ Python จะไม่มีตัวแปร argc เนื่องจาก Python สามารถหาค่าจำนวนตัวแปรได้จากตัวแปร argv แทน โดยใช้คำสั่ง len(sys.argv) สำหรับการสั่งรัน Python ผ่านทาง MS-DOS command line ใช้รูปแบบคำสั่งดังนี้คือ <python> <ชื่อโปรแกรม.py><พารามิตเตอร์ 1><พารามิตเตอร์ 1><พารามิตเตอร์ N> เช่น

```
C:>python Lab1.py para1 para2 ↵
```

สำหรับรายละเอียดเพิ่มเติมสามารถอ่านได้ในบทที่ 3

การส่งค่า Arguments ผ่านทาง Command-Line (Parsing Command-Line Arguments)

Python ได้จัดเตรียมคลาสชื่อว่า getopt เพื่อใช้สำหรับช่วยในการส่งค่า options และ arguments ให้กับโปรแกรมเพื่อทำงาน โดยค่าเหล่านี้ถูกส่งไปพร้อมกับขณะกำลังสั่งรันโปรแกรม ซึ่งมีรูปแบบดังนี้

```
getopt.getopt(args, options[, long_options])
```

โดย args คือ argument ที่ต้องการส่งไปให้โปรแกรมทำงาน

Options คือ options ที่ต้องการสั่งให้โปรแกรมทำงาน โดยแยกแต่ละ option ด้วยเครื่องหมาย : เช่น hi:o

Long options เป็นค่าที่ใช้ หรือไม่ใช้งานก็ได้ แต่ถ้าใช้งานจะต้องเป็นตัวแปรชนิดลิสต์ และในตัวแปรจะต้องมีข้อมูลที่เป็น option แบบยาว โดย option จะต้องมีตัวจักรกกำหนดค่าโดยใช้ เครื่องหมาย = เช่น ["ifile=","ofile="]

ตัวอย่างการใช้งาน

ผู้เขียนโปรแกรมต้องการสั่งชื่อไฟล์ 2 ชื่อ พร้อมกับสั่งให้โปรแกรมทำงานทาง command-line โดยผู้เขียนต้องการให้โปรแกรมสามารถทำการตรวจสอบ options ที่ส่งไปพร้อมกับโปรแกรมได้ ซึ่งรูปแบบการเรียกใช้โปรแกรม มีรูปแบบคือ

```
usage: test.py -i <inputfile> -o <outputfile>
```

ตัวอย่างโปรแกรม test.py

```
import sys, getopt  
def main(argv):  
    inputfile = ''
```

```

outfile = ''
try:
    opts, args = getopt.getopt(argv,"hi:o:",["ifile=","ofile"])
except getopt.GetoptError:
    print ('test.py -i <inputfile> -o <outputfile>')
    sys.exit(2)
for opt, arg in opts:
    if opt == '-h':
        print ('test.py -i <inputfile> -o <outputfile>')
        sys.exit()
    elif opt in ("-i", "--ifile"):
        inputfile = arg
    elif opt in ("-o", "--ofile"):
        outfile = arg
print ('Input file is "', inputfile)
print ('Output file is "', outfile)

if __name__ == "__main__":
    main(sys.argv[1:])

```

ทดสอบสิ่งให้โปรแกรมทำงานโดยเรียกคำสั่งดังต่อไปนี้ใน command-line

```

C:> python test.py -h
usage: test.py -i <inputfile> -o <outputfile>

C:> python test.py -i BMP -o
usage: test.py -i <inputfile> -o <outputfile>

C:> python test.py -i inputfile
Input file is " inputfile
Output file is "

```

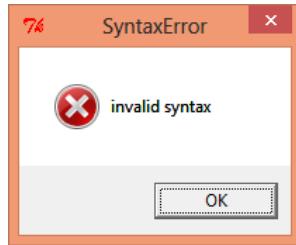
การตรวจสอบความผิดพลาดของโปรแกรม

เป็นที่ทราบกันดีว่า เมื่อได้ก็ตามที่เริ่มเขียนโปรแกรมจะมีข้อผิดพลาด (Bug) ตามมาทันที เมื่อมองกับเราติดตามด้วย (เป็นธรรมชาติ เพราะมนุษย์นั้นผิดพลาดอยู่เสมอ) เราจำเป็นต้องมีตัวช่วยในการค้นหาความผิดพลาดเหล่านั้น ในภาษา Python ได้เตรียมเครื่องมือเหล่านี้ไว้ให้เราเรียบร้อยแล้ว เช่นเดียวกัน ในหัวข้อนี้ผู้เขียนจะแนะนำการใช้เครื่องมือเหล่านั้น เพื่อใช้เป็นเครื่องมือในการตรวจสอบความผิดพลาดที่เกิดจากการเขียนโปรแกรมของเราเอง

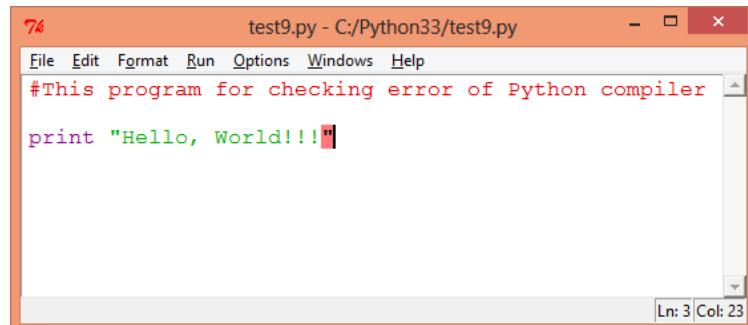
อันดับแรกขอเริ่มจากการตรวจสอบความผิดพลาดที่ไฟล์อนุคอมไฟเลอร์ตรวจสอบให้ก่อน ตัวอย่างเช่น เมื่อเราเขียนโปรแกรมดังต่อไปนี้

```
#This program for testing errors of Python compiler
print "Hello, World!!!!"
```

เมื่อสั่งรันโปรแกรม จะแสดงข้อความเตือนว่า invalid syntax



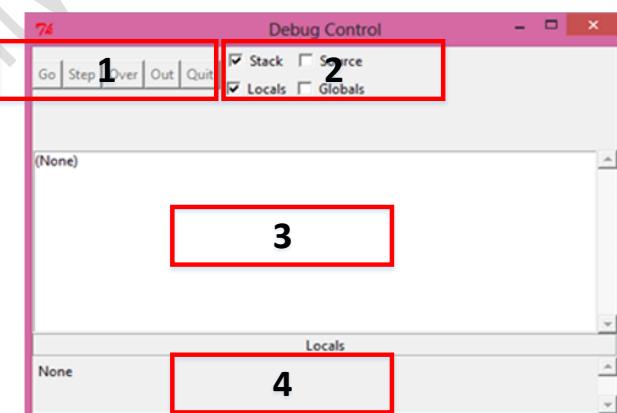
และเคอเซอร์ หรือพรอมพท (prompt) จะมาปรากฏอยู่ในตำแหน่งที่ผิดพลาดทันที



ซึ่งแสดงให้ผู้เขียนโปรแกรมเห็นว่า ไวยกรณ์ได้เขียนผิดพลาดแล้ว ให้แก้เสียใหม่ ถึงตรงจุดนี้ก็ขึ้นอยู่กับว่า ผู้เขียนมีความชำนาญในการเขียนโปรแกรมมากน้อยเพียงใด ที่จะแก้ปัญหาความผิดพลาดนี้ให้หมดไป จากโปรแกรมข้างต้น ให้ผู้อ่านแก้ไขจาก `print "Hello, World!!!"` เป็น `print ("Hello, World!!!")` แทนเนื่องจาก Python 3 ได้ยกเลิกการใช้งาน `print` แบบ "... " ของเวอร์ชันต่ำกว่า 3.0 ไปแล้ว

การตรวจสอบโปรแกรมโดยใช้ Debug

เริ่มต้นที่โปรแกรม Python shell ให้ผู้ใช้เลือกเมนู File \Rightarrow Open \Rightarrow เลือกโปรแกรมที่ต้องการตรวจสอบ .py \Rightarrow เลือก Debug \Rightarrow Debugger จะปรากฏ Debug control ดังรูปที่ 2.12



รูปที่ 2.12 Debug Control

ในหน้าต่างของ Python shell จะปรากฏข้อความว่า [DEBUG ON] และว่าไฟช่อน Debugger ได้เริ่มทำงานแล้ว ในเมนูของ Bebug Control จะมีส่วนประกอบสำคัญๆ คือ

1. ปุ่มควบคุมการสั่งรันโปรแกรม

Go สั่งให้โปรแกรมทำงานทั้งหมดทันที

Step สั่งให้โปรแกรมทำงานที่ลับบรรทัด โดยสามารถดูข้อมูลได้อย่างละเอียด แสดงทุกตัวแปรในโปรแกรม รวมถึงตัวแปรของระบบที่ซ่อนอยู่ด้วย

Over สั่งให้โปรแกรมทำงานที่ลับบรรทัด เฉพาะที่อยู่ในโค้ดของผู้เขียน โปรแกรมเท่านั้น (ไม่รวมตัวแปรของระบบ)

Out สั่งให้โปรแกรมทำงานเฉพาะส่วนที่แสดงผลลัพธ์ออกทางจอภาพเท่านั้น

Quit ออกจากโหมดการ Debug

2. ปุ่มควบคุมการแสดงผลตัวแปร

Stack แสดงข้อมูลหน่วยความจำใน Stack

Locals แสดงข้อมูลของตัวแปรชนิด local

Source แสดงโปรแกรมแฟ้มต้นฉบับพร้อมกันขณะทำการ Debug

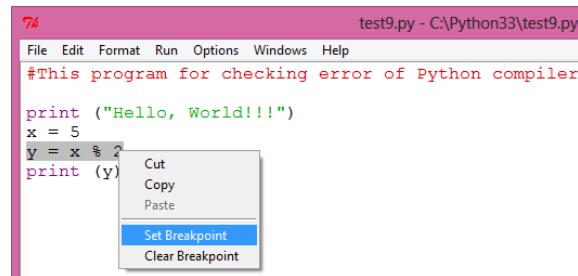
Globals แสดงข้อมูลของตัวแปรชนิด global

3. แสดงข้อมูลที่อยู่ใน stack และคำสั่งต่างๆ ขณะทำการ Debug

4. แสดงข้อมูลที่เก็บในตัวแปรชนิด local

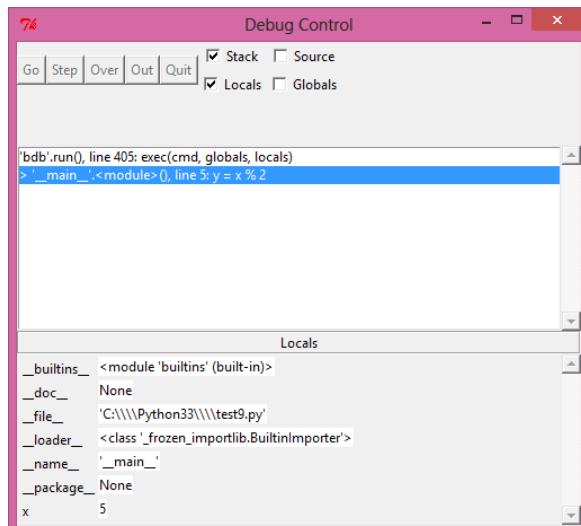
การตรวจสอบแบบใช้ Break point

เมื่อโปรแกรมที่ถูกเขียนขึ้นมีจำนวนบรรทัดมากขึ้นเรื่อยๆ การที่ Debug ตั้งแต่บรรทัดที่ 1 คงจะไม่สะดวกเท่าที่ควร ดังนั้นไฟช่อนได้เตรียม Break point ไว้ให้เราด้วย ซึ่งผู้เขียนโปรแกรมสามารถเลือกบรรทัดใดก็ได้ที่ต้องการตรวจสอบ (โดยการคลิกขวาใน Python editor และเลือก Set breakpoint ถ้าต้องการยกเลิกให้เลือก Clear breakpoint) เมื่อโปรแกรมแปลภาษาทำงานจะมาหยุดตรงที่เราทำเครื่องหมายไว้ เพื่อรอให้ผู้ใช้สามารถดูรายละเอียดของตัวแปร และผลลัพธ์ได้ทันที เช่น ดังรูปที่ 2.13



รูปที่ 2.13 แสดงการกำหนด Breakpoint

ผลลัพธ์ที่ได้แสดงดังรูปที่ 2.14



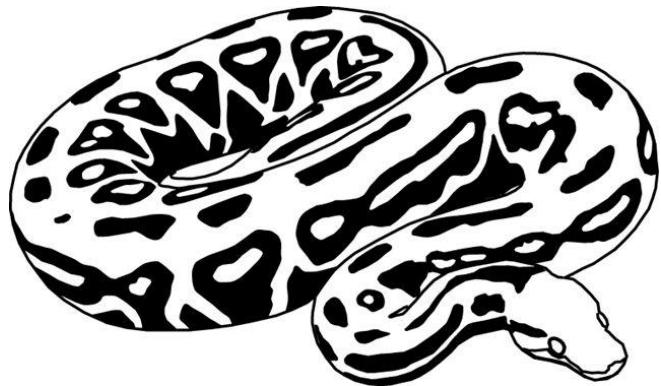
รูปที่ 2.14 แสดงผลการรันเมื่อโปรแกรมมาหยุดที่จุด breakpoint

สรุปการใช้งาน Python Debugger

1. เริ่มต้นจาก Python shell \Rightarrow Open \Rightarrow เลือกโปรแกรมต้นฉบับ.py \Rightarrow Python shell \Rightarrow Debugger
2. หน้าต่าง Python editor \Rightarrow คลิกขวาเลือกบรรทัดที่ต้องการทำ Breakpoint \Rightarrow กดปุ่ม F5 เพื่อรันโปรแกรม
3. ตรวจสอบข้อมูล ตัวแปร หน่วยความจำใน Debug Control โดยกดปุ่ม Go หรือ Step เพื่อให้โปรแกรมทำงานไปเรื่อยๆ

จบบทที่ 2

จบภาค 1

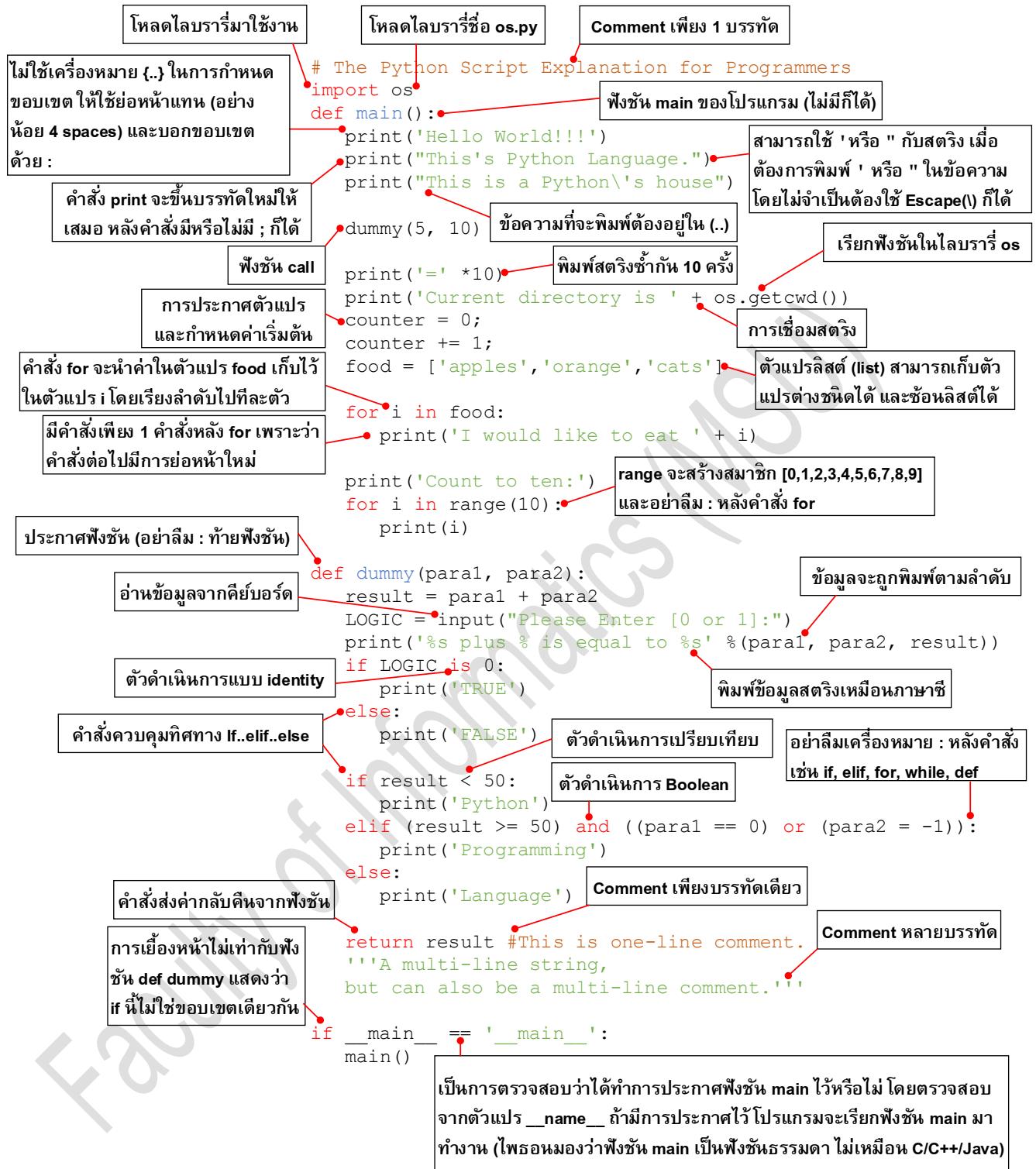


ภาคที่ 2

การเขียนโปรแกรมภาษาไพธอนขั้นพื้นฐาน



- บทที่ 3 โครงสร้างการเขียนโปรแกรมภาษาไพธอน
- บทที่ 4 ตัวแปร การกำหนดค่าและชนิดข้อมูล
- บทที่ 5 นิพจน์ ตัวดำเนินการ และตัวถูกดำเนินการ
- บทที่ 6 เสื่อนไข การตัดสินใจ การควบคุมทิศทาง และการทำซ้ำ
- บทที่ 7 พัฒนา (Functions)
- บทที่ 8 โมดูลและแพ็คเกจ
- บทที่ 9 การจัดการข้อผิดพลาด
- บทที่ 10 การจัดการไฟล์ข้อมูล
- บทที่ 11 การเขียนโปรแกรมเชิงวัตถุ



ภาพรวมการเขียนโปรแกรมภาษาไพธอนสำหรับโปรแกรมเมอร์

บทที่ 3

โครงสร้างการเขียนโปรแกรมภาษาไพธอน

(Python programming structure)



เกือบกalty เป็นธรรมเนียมปฏิบัติไปแล้ว เมื่อกล่าวถึงหนังสือ หรือตำราที่บรรยายถึงการเขียนโปรแกรม ไม่ว่าจะเป็นภาษาใด ๆ ก็ตาม เกือบทั้งหมด โปรแกรมแรกจะต้องเริ่มต้นด้วย “สวัสดีชาวโลก” หรือ “Hello, World” ซึ่งถ้าไม่คิดอะไรมากก็ดูเรียบง่ายดี แต่ในมุมมองของผู้เขียนมันมีนัยสำคัญแยะอยู่ หลายประการ ประการแรก ผู้เขียนโปรแกรมที่เป็นมือใหม่ (หัดขับ...หัดเขียน) ยังไม่มีทักษะในการเขียนโปรแกรมมาเลย ซึ่งแน่นอนมันเป็นการเรียนรู้ภาษาใหม่ ต้องใช้เวลา ถ้าจะให้เขียนยาก ๆ ต้องมีการตรวจสอบโน่นตรวจสอบนี่ ผู้เขียนรับรองว่าได้เลิกก่อนอย่างแน่นอน แต่ถ้าสามารถเขียนโปรแกรมแรกได้โดยไม่มีข้อผิดพลาดใด ๆ ก็ได้ขึ้นเลย กำลังใจที่จะเขียนโปรแกรมก็จะตามมาด้วย ประการที่สอง โดยปกติเมื่อติดตั้งโปรแกรมแปลภาษาใหม่ลงในเครื่องคอมพิวเตอร์ จำเป็นต้องมีการทดสอบก่อนว่าภาษาที่ติดตั้งลงไปใหม่นั้นสามารถทำงานได้หรือไม่ ซึ่งวิธีการทดสอบโดยพื้นฐานก็คือ เขียนโปรแกรมด้วยไวยกรณ์ของภาษาดังกล่าว และให้มันทำอะไรสักอย่างที่ง่าย ๆ และสะทกสะทุกดสอบ ซึ่งก็คือการใช้คำสั่งการพิมพ์ข้อความออกจากภาพหนึ่งเอง ซึ่งต่อมาก็จะกล่าวเป็นคำาณขึ้นอีกว่า และจะพิมพ์อะไรดี ล่ะ? สรุปสุดท้ายก็มาลงที่ สวัสดีชาวโลก หรือ Hello, World นี่แหล่งง่ายดี และประการสุดท้ายเหมือนจะเป็นวัฒธรรมที่ฝัง根柢ในจิตใจของทุก ๆ คนบนโลกนี้ไปแล้ว เมื่อเราพบเจอกันในครั้งแรก จะต้องมีการทักทายกันก่อนเสมอ เช่น สวัสดี Hello เป็นต้น ดังนั้นการใช้คำสำคัญ (Keyword) ว่า สวัสดีชาวโลก หรือ Hello, World ก็เปรียบเสมือนเป็นการทักทายกันระหว่างภาษาคอมพิวเตอร์กับผู้เขียนโปรแกรม หน้าใหม่ที่เพิ่งเจอกันเป็นครั้งแรกเช่นเดียวกัน เพื่อคงธรรณะนิยมอันดีนี้ไว้ ผู้เขียนจะขอเริ่มต้นด้วยการทักทายแบบเดียวกัน แต่จะทำให้โปรแกรมแรกของเรานั้นลาดขึ้นมากหน่อย โดยมีการโต้ตอบกับผู้เขียน โปรแกรมด้วย ดังตัวอย่างต่อไปนี้

Output Example 3.1: Hello World!	
---	--

Output Example 3.1: Hello World!	
1	Computer said : Hello, I'm Python!!!
2	Computer said : How are you today? Suchart
⇒3	You said : I'm good ↳ Enter your feeling
4	Computer said : Ohh! you are good !!
5	You said : How about you?
6	Computer said : Very good!

จากເອົາຕົວໃນໂປຣແກຣມຕ້ວຍຢ່າງ (Output Example 3.1) ທີ່ 3.1 ແສດງໄທ້ເຫັນກາຣຕອບໂຕຮ່ວງຜູ້ເຂີຍໂປຣແກຣມແລະຄອມພິວເຕອີ່ ໂດຍໃນບຣທັດທີ່ 1 ໂປຣແກຣມຈະແສດງກາຣທັກທາຍກ່ອນດ້ວຍປະໂຍຄວ່າ “Hello, I'm Python!!!” ຈາກນັ້ນບຣທັດທີ່ 2 ໂປຣແກຣມຈະຄາມຜູ້ເຂີຍໂປຣແກຣມວ່າ ສນາຍດີແໜ່ວນນີ້ ພຣ້ອມດ້ວຍຊື່ອຂອງຜູ້ເຂີຍໂປຣແກຣມ (“How are you today? Suchart”) ໃນບຣທັດທີ່ 3 ຜູ້ເຂີຍໂປຣແກຣມຕ້ອງແສດງຄວາມຮູ້ສຶກເພື່ອເປັນກາຣທັກທາຍກັບຄອມພິວເຕອີ່ຢ່າງໄດ້ຍ່າງໜຶ່ງເຂົ້າປີ ເຊັ່ນ Good, bad, so so ເປັນດັນບຣທັດທີ່ 4 ຄອມພິວເຕອີ່ຈະແສດງຄວາມຮູ້ສຶກໄປກັບອາຮມັນຂອງຜູ້ເຂີຍໂປຣແກຣມດ້ວຍປະໂຍຄວ່າ “Ohh! you are your feeling !!” ບຣທັດທີ່ 4 ຜູ້ເຂີຍໂປຣແກຣມຈະຄາມກັບຄອມພິວເຕອີ່ບ້າງວ່າເປັນຍ່າງໄວ “How about you?” ບຣທັດທີ່ 5 ຄອມພິວເຕອີ່ຕອບວ່າ “Very good!” ໂປຣແກຣມຈຶ່ງຈະການທຳການ

1. ສວັສດີ ລັ້ນຄືອີ່ໄພຮອນ Hello, I'm Python

ໃຫ້ຜູ້ອ່ານທຸລອງເຂີຍໂປຣແກຣມດັ່ງຕ້ວຍຢ່າງດ້ານລ່າງ ບັນທຶກໄຟລ໌ຊື່ອ Hello.py ເສົ້າງແລ້ວໃຫ້ກົດສອບຮັນໂປຣແກຣມ ຜ່ານກາງ Command line (MS-DOS) ດັ່ງນີ້

C:\Python33> python Hello.py ຂໍອັນຫຼັກໂປຣແກຣມ
ເຊັ່ນ C:\Python33> python Hello.py Suchart

Program Example 3.1: Hello World!

```
1 #!/usr/bin/python
2 #This program looks like the expert system, you can talk with
3 #this computer.
4 #How to run this program:
5 #In MS-DOS prompt, you should type: python name.py your_name
6 #For example : python exam3_1.py Suchart
7 #Version 1.0
8 #Written by Mr Suchart
9 #GNU License : You can modify and publish this software
10 #everywhere
11
12 import sys
13
14 if len(sys.argv) == 2 :
15     if sys.argv[1] != '' :
16         print("Computer said : Hello, I'm Python!!!!")
17         print("Computer said : How are you today?")
18         print(sys.argv[1])
19         feeling = input("You said : I'm ")
20         print("Computer said : Ohh! you are ",feeling,"!!")
21         print("You said : How about you?")
22         print("Computer said : Very good!")
23
24 else :
25     print("Computer said : I cannot talk with you, because
your argv was missing!!!!")
```

โปรแกรม Hello.py มีลักษณะการทำงานคล้ายระบบผู้ใช้ภาษาญี่ปุ่นเช่นภาษาคอมพิวเตอร์ได้ ซึ่งโปรแกรมแรกของเรา แม้ว่าจะโต้ตอบกับคอมพิวเตอร์ได้ไม่มากนัก แต่ก็ถือว่าเป็นการเริ่มต้นในการเขียนโปรแกรมแรกที่ดูท้าทายขึ้น ต่อไปเรามาลองศึกษา กันดูว่าแต่ละบรรทัดหมายถึงอะไร

บรรทัดที่ 1: เครื่องหมาย # คือการ Comment หมายความว่าตัวเปล่งภาษาจะข้ามบรรทัด ดังกล่าวไป โดยไม่เปลี่ยนความหมายใดๆ แต่ สำหรับในบรรทัดนี้ มีความหมายที่แตกต่างไปเล็กน้อยคือ หลังเครื่องหมาย # มีข้อความ !/usr/bin/python ซึ่งหมายถึงการประกาศให้โปรแกรมรู้ว่าเรียกตัวเปล่งภาษาได้จากที่ไหน สำหรับในที่นี้เป็นการประกาศบนระบบปฏิบัติการยูนิกซ์-ลินุกซ์ สำหรับวินโดวส์ สามารถลบบรรทัดดังกล่าวทิ้งไปได้เลย ไม่จำเป็นต้องใช้งาน

บรรทัดที่ 2 - 8: คือการอธิบายว่าโปรแกรมทำงานอะไร ใช้ข้อมูลอะไรบ้างเพื่อให้โปรแกรมทำงานได้สมบูรณ์ โปรแกรมใช้งานได้อย่างไร เวอร์ชันอะไร ใครเป็นผู้พัฒนาโปรแกรม รวมถึงอาจจะกล่าวถึงเรื่องลิขสิทธิ์ด้วยถ้าจำเป็น

บรรทัดที่ 10: คือการเรียกใช้คลาส sys (system) เป็นไลบรารีที่จัดการเกี่ยวกับระบบ (System)

บรรทัดที่ 12: คือการตรวจสอบเงื่อนไขว่าพารามิเตอร์ที่รับเข้ามาพร้อมกับโปรแกรมนั้น มีจำนวน 2 ตัวหรือไม่ (len(sys.argv) คือการหาจำนวนพารามิเตอร์) คือ ชื่อโปรแกรม (Hello.py) และชื่อผู้เขียนโปรแกรม (Suchart) ถ้ามีจำนวนตัวแปรเท่ากับ 2 ให้ทำการสั่งหลัง if ต่อไป ถ้าไม่เป็นจริง ซึ่งเปลี่ยนความหมายได้ 2 กรณี คือ ขณะสั่งรันโปรแกรมไม่ใส่พารามิเตอร์ต่อท้ายไฟล์มาด้วย หรือใส่พารามิเตอร์เกินที่กำหนด โปรแกรมจะกระโดดไปทำงานในบรรทัดที่ 21 (หลังคำสั่ง else) เพื่อพิมพ์ข้อความว่า "I cannot talk with you, because your argv was missing!!!" และจบการทำงานทันที

บรรทัดที่ 13: คือการตรวจสอบเงื่อนไขว่าชื่อที่ต่อท้ายไฟล์ขณะทำการรันโปรแกรมนั้น เป็นค่า ว่างหรือไม่ (ผู้ใช้งานอาจจะใส่ค่า Space) ถ้าเป็นค่าว่าง โปรแกรมจะจบการทำงานทันที แต่ถ้าเป็นจริง จะทำงานในบรรทัดที่ 14 ต่อไป

บรรทัดที่ 14: คอมพิวเตอร์จะพิมพ์ข้อความแนะนำตัวเอง "Hello, I'm Python!!!" กับผู้ใช้งาน

บรรทัดที่ 15: คอมพิวเตอร์จะพิมพ์ข้อความทักทายผู้ใช้ว่า "How are you today?" พร้อมกับชื่อของผู้ใช้งานอ กมา ซึ่งชื่อของผู้ใช้งานจะรับเข้ามาตอนสั่งรันโปรแกรม โดยเก็บอยู่ในตัวแปร argv[1]

บรรทัดที่ 16: คอมพิวเตอร์จะรับข้อความจากคีย์บอร์ดของผู้ใช้ว่าตอนนี้มีความรู้สึกเป็นอย่างไรบ้าง ให้ผู้ใช้ป้อนคำตอบเป็นความรู้สึกขณะนี้ว่าเป็นอย่างไร เช่น Very good, Good, So so, Bad เป็นต้น

บรรทัดที่ 17: คอมพิวเตอร์จะพิมพ์ข้อความแสดงความรู้สึกกับผู้ใช้งาน เช่น Ohh! you are good !!

บรรทัดที่ 18: ตามมาเรียกคุณควรตอบกลับไปว่า “How about you?”

บรรทัดที่ 19: คอมพิวเตอร์จะตอบกลับมาว่า “Very good!” และจบการทำงาน

2. โครงสร้างการเขียนโปรแกรมไพธอน (Python programming structure)

โดยปกติของภาษาโปรแกรมมิ่งทั่วๆ ไป ทุกๆ โปรแกรมจะมีฟังชันหลักที่เรียกว่า Main function เสมอ ยกตัวอย่างในโปรแกรมภาษา C ต่อไปนี้

```
void doit ( int x ) { x = 5; }
int main() {
    int z = 27;
    doit(z);
    fprintf('z is now %d\n', z);
    return 0; }
```

จากตัวอย่างโปรแกรมภาษา C จะมีฟังชัน main เป็นฟังชันที่ควบคุมการทำงานของคำสั่ง และ พังชันย่อยอื่นๆ ในโปรแกรมเสมอ แต่สำหรับไพธอนไม่จำเป็นต้องมีฟังชัน main ก็ได้ แต่ถ้าผู้เขียน โปรแกรมต้องการใช้งานฟังชัน main ก็สามารถทำได้ แต่ไพธอนมองว่าฟังชัน main เป็นเพียงฟังชัน ทั่วๆ ไป ไม่ได้มีความหมายเหมือนอย่างในภาษาการดับสูงอื่นๆ เช่น C/C++ หรือ Java เป็นต้น

โครงสร้างภาษาการเขียนโปรแกรมไพธอน

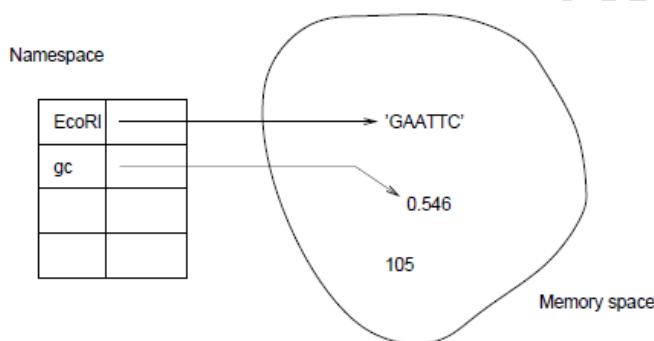
ขอบเขต	ตัวอย่างโปรแกรมไพธอน	ความหมาย
1	<code>#!/usr/bin/python</code>	คอมเมนต์ หรือการประกาศตัวแปรภาษา
2	<code>import sys, getopt</code>	นำเข้าไลบรารี หรือคลาสของไพธอนมาใช้งาน
3	<code>def display(): print("Python programming") def main(): print("I'm the Main function") display() if __name__ == "__main__": main()</code>	ประกาศตัวแปร ฟังชัน และคำสั่งควบคุมต่างๆ รวมถึงฟังชัน main ด้วย

3. ไวยกรณ์พื้นฐานที่จำเป็นอย่างยิ่งต้องจดจำ (Important basic syntax)

ไวยกรณ์ต่างๆ ที่จะกล่าวต่อไปนี้ ขอให้ผู้อ่านจดจำ และท่องให้ขึ้นใจ เพราะมันจะทำให้การเขียนโปรแกรมไม่มีอุปสรรค

1. Case sensitivity: การตั้งชื่อตัวแปร ตัวใหญ่ และตัวเล็กถือว่าเป็นคนละตัวแปร เช่น Number และ number ไม่ใช้ตัวแปรตัวเดียวกัน
2. Space and tabs don't mix: ทราบมองว่า space และ tabs มีความหมายไม่เหมือนกัน ดังนั้นเวลาเขียนโปรแกรม อย่าผสมระหว่าง space และ tabs เข้าด้วยกันให้เลือกเอาอย่างใดอย่างหนึ่งเท่านั้น
3. Objects (วัตถุ หรือเรียกทับศัพท์ว่าอ็อปเจกต์): ไฟรอนถูกสร้างขึ้นภายใต้แนวคิดการโปรแกรมเชิงวัตถุ ดังนั้นเมื่อเราเรียกใช้งานคลาสใดๆ ก็ตามถือว่าเป็นวัตถุตามแนวความคิดแบบโปรแกรมเชิงวัตถุ (การโปรแกรมเชิงวัตถุจะกล่าวในบทที่ 11) ดังนั้น เมื่อได้ก็ตามที่มีการสร้างวัตถุและต้องการเข้าถึงแอทริบิวต์ (Attribute) หรือฟังชัน (Function) ไดๆ ในวัตถุต้องใช้ “.” และตามด้วยเครื่องหมาย () แต่ถ้าอ้างถึงตัวแปรไม่ต้องมี () เช่น เมื่อต้องการเปลี่ยนค่าสตริง ‘Pop’ เป็นตัวอักษรตัวใหญ่ทั้งหมด ทำได้โดยเรียกใช้คลาส upper ในไลบรารีของไฟรอน ดังนี้ ‘Pop’.upper()
4. Scope: ในการพัฒนาโปรแกรมขนาดใหญ่ ที่มีโปรแกรมเมอร์มากกว่า 1 คน อาจจะประสบปัญหาเรื่องการประกาศตัวแปรที่ซ้ำกันได้ ดังนั้นเพื่อให้การเข้าถึงและใช้งานตัวแปรเป็นไปอย่างถูกต้องโดยไม่มีข้อผิดพลาด แนะนำให้ผู้เขียนโปรแกรมใช้งานในลักษณะของฟังชันจะดีกว่า โดยมีการส่งค่าตัวแปรไปในฟังชัน และคืนค่าที่คำนวนเรียบร้อยแล้วกลับมา จะไม่ทำให้ประสบปัญหารื่องของการอ้างตัวแปรดังที่กล่าวมาแล้ว
5. Namespaces: คือพื้นที่ที่ใช้เก็บตัวแปรของระบบที่สร้างไว้ให้เราโดยที่เราไม่รู้ และตัวแปรต่างๆ ที่เราสร้างขึ้นมาที่หลัง เราสามารถขอดูข้อมูลที่เก็บอยู่ใน Namespaces (รูปที่ 3.1) ได้โดยใช้คำสั่ง dir() ซึ่งเป็น built-in function ที่มีอยู่ในไฟรอน ซึ่งถ้าเรายังไม่ได้ประกาศตัวแปร หรือฟังชันใดๆ ในโปรแกรมจะปรากฏรายการของตัวแปรที่ระบบสร้างไว้ให้ 6 ตัวคือ '__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__' ถ้าต้องการดูประเภทของตัวแปรเหล่านี้ว่าเป็นชนิดอะไร สามารถเรียกดูได้โดยใช้คำสั่ง type() เช่น type(__builtins__) ชนิดของตัวแปรที่ปราภูมิคือ <class 'module'> หรือ type(__doc__) ชนิดของตัวแปรที่ปราภูมิคือ <class 'NoneType'> เมื่อตัวแปรเป็นชนิด module เราสามารถดูข้อมูลภายในโมดูลเหล่านั้นได้ด้วยคำสั่ง dir() เช่น dir(__builtins__) ผลที่ได้คือชื่อของฟังชัน หรือคลาสที่อยู่ภายใต้ห้องแม่ดู กما ดังนี้ ['ArithmeticError', 'AssertionError', 'AttributeError', ..., 'zip'] ในแต่ละ Namespaces เปรียบเสมือนเป็นคลัง

หรือตู้คอนเทนเนอร์สำหรับเก็บข้อมูลต่างๆ ลงไป ดังนั้นแต่ละโปรแกรมจะถูกเก็บแยกออกจากกันโดยอิสระ ดังนั้นอาจจะเรียก Namespaces ว่าเหมือนกับ Scope ก็ได้ เมื่อใดก็ตามที่เราสร้างตัวแปร หรือฟังชัน ตัวแปรที่สร้างขึ้นก็จะถูกเก็บอยู่ในพื้นที่ของ Namespaces ทั้งหมด เมื่อเราทำการนำเข้าคลาสใดๆ เข้ามาใช้งานในโปรแกรมด้วยการใช้ import คลาส ต่างๆ เหล่านั้นก็จะมาปรากฏใน Namespaces ด้วย เช่น import math จากนั้นใช้คำสั่ง dir() ผลลัพธ์ที่ได้คือ ['__builtins__', '__doc__', '__loader__', '__name__', '__package__', 'math', 'x'] ถ้าผู้เขียนโปรแกรมต้องการทราบรายละเอียดของฟังชัน หรือเมธอด (Method) ในคลาส math ให้ใช้คำสั่ง dir(math) ถ้าผู้เขียนโปรแกรมต้องการทราบการทำงานในแต่ละฟังชันของ math ว่าทำงานอย่างไร สามารถทำได้โดยใช้ฟังชัน print ตามด้วยชื่อคลาส.เมธอด เช่น print (math.pow) หรือ print (math.pi) เป็นต้น



รูปที่ 3.1 แสดง Namespace

6. Colons: "เพื่อนตัดเครื่องหมายแสดงขอบเขตของข้อมูล {...} ทิ้งไป แล้วใช้ : ร่วมกับการเขียนโปรแกรมด้วยการย่อหน้าแทน โดยเริ่มจากคลัมก์ที่ 1 เสมอดังนี้อย่าลืม : หลังคำสั่ง if, for, while, def เป็นอันขาด
7. Blank lines: เมื่อจำเป็นต้องเขียนคำสั่งที่มีความยาวมากๆ ไม่หมดใน 1บรรทัด ให้ใช้เครื่องหมาย \ ตามด้วย enter เช่น

```
print ('This is a really long lines \
but we can make it across \
multiple lines')
```

หรือ

```
x = 4 * 5 - 5 + \
    6 + 8 \
    + 5 % 2
print(x)
```

8. Lines and Indentation: ไฟรอนไม่ใช้เครื่องหมาย {...} ในการกำหนดขอบเขต เมื่อในภาษาซี ไฟรอนใช้การยื่ง หรือย่อหน้าแทน ดังนั้นผู้เขียนโปรแกรมจะต้องระวังการยื่งหน้าให้ดี จากตัวอย่างต่อไปนี้

ตัวอย่างที่ 1:	ตัวอย่างที่ 2:
<pre>if True: print ("True") else: print ("Answer") print ("False")</pre>	<pre>if True: print ("Answer") print ("True") else: print ("Answer") print ("False")</pre>
โปรแกรมตัวอย่างที่ 1 จะไม่เกิดข้อผิดพลาด เพราะว่าคำสั่งหลัง else ย่อหน้าตรงกัน	โปรแกรมตัวอย่างที่ 2 จะเกิดข้อผิดพลาด เพราะว่าคำสั่งหลัง else ย่อหน้าไม่ตรงกัน

9. Multi-line statements: แต่ละคำสั่งของไฟรอนส่วนใหญ่จบลงด้วยการขึ้นบรรทัดใหม่ (new line) แต่ผู้เขียนโปรแกรมสามารถใช้เครื่องหมาย \ เพื่อเชื่อมคำสั่งได้ เช่น

```
total = item_one + \
        item_two + \
        item_three
```



one two three

OUTPUT

สำหรับข้อมูลในเครื่องหมาย [...], [...] หรือ (...) ไม่จำเป็นต้องใช้เครื่องหมาย \ เช่น

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

10. Quotation in Python: ไฟรอนใช้เครื่องหมาย ' (single quote), " (double quote) ในการแสดงค่าของสตริง แต่เครื่องหมาย """ (triple quote) สามารถใช้เชื่อมต่อสตริงแบบหลาย ๆ บรรทัดได้ เช่น

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
               made up of multiple lines and
               sentences."""
print(paragraph)
```



This is a paragraph. It is
made up of multiple lines and sentences.

OUTPUT

11. Waiting for the user: บอยครั้งที่ผู้เขียนโปรแกรมต้องการให้โปรแกรมหยุดรอ ก่อนโปรแกรมทำงานเสร็จ โดยขึ้นข้อความว่า “Press the enter key to exit.” สามารถใช้ `\n` แทน ใส่ไว้ก่อนข้อความ ดังนี้

```
input("\n\nPress the enter key to exit.")
```

12. Multiple statements on a single line: ผู้เขียนโปรแกรมสามารถใช้เครื่องหมาย ; เพื่อสั่งให้สามารถรันหลาย ๆ คำสั่งได้ในบรรทัดเดียวกันได้

```
import sys; x = 'Python'; sys.stdout.write(x + '\n')
```

 OUTPUT	Python 7
--	-------------

4. คำสั่งการแสดงผล (Print command)

ผู้ที่เริ่มต้นฝึกการเขียนโปรแกรม คำสั่งแรกที่ควรจะทำความเข้าใจ คือคำสั่ง การพิมพ์ข้อมูลออกทางจอภาพ (print) เพราะเป็นคำสั่งที่ช่วยทดสอบผลลัพธ์ในการประมวลผลของโปรแกรม คำสั่ง print เป็นฟังชันชนิด built-in ซึ่งทำหน้าที่ 2 อย่างคือ

- 1) ทำหน้าที่แปลง Object ไปเป็นสตริง และ
- 2) พิมพ์สตริงออกทางจอภาพ (Standard output)

ทดสอบฟังชัน print

```
print(); #สั่งให้ขึ้นบรรทัดใหม่ (new line หรือ \n)
```

สตริงในภาษาไพธอนจะอยู่ภายใต้เครื่องหมาย '' หรือ "" ซึ่งสามารถใช้แทนกันได้ และใช้งานได้ยืดหยุ่น เช่น ถ้าผู้เขียนโปรแกรมต้องการพิมพ์ข้อความที่มี ' อัญญิสตริง ให้ผู้เขียนใช้ " ครอบสตริง อัญญิตัวนอกแทน (Nesting Quotes) เช่น ต้องการพิมพ์ข้อความว่า Hi Dad, Isn't it lovely? และ I said, "Hi" สามารถสั่งพิมพ์คือ

```
print ("Hi Dad", "Isn't it lovely?", ' I said, "Hi".')
```

 OUTPUT	Hi Dad Isn't it lovely? I said, "Hi".
--	---------------------------------------

เมื่อต้องการพิมพ์ข้อความเดิมซ้ำๆ และติดกัน สามารถใช้สัญลักษณ์ * (Repetition Symbol) ได้

```
print ("Hello!"*5)
```

 OUTPUT	Hello!Hello!Hello!Hello!Hello!
--	--------------------------------

ถ้าต้องการพิมพ์ข้อมูลแบบหลายบรรทัดพร้อมๆ กัน ให้ใช้เครื่องหมาย ; ร่วมกับคำสั่ง print

```
print ("this should be");
print ("on the same line");
```



this should be
on the same line

OUTPUT

พิมพ์ข้อมูลในตัวแปรที่เก็บข้อมูลสตริง

```
String = "Python programming."
print ('My subject is ',String)
```



My subject is Python programming.

OUTPUT

เชื่อมต่อสตริงเข้าด้วยกันใช้สัญลักษณ์ + สำหรับเชื่อม

```
String = 'My subject is ' + "Python programming" + "language"
print ('My subject is ',String)
```



My subject is Python programming language

OUTPUT

ต้องการพิมพ์ข้อความที่มีความยาวมากกว่า 1 บรรทัดให้ใช้เครื่องหมาย \ (escape) เมื่อ

ต้องการขึ้นหน้าใหม่สามารถใช้ \n ได้ คล้ายภาษา C เช่น

```
Hello = ("This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
Note that whitespace at the beginning of the line is\
significant.")
print (Hello)
```



This is a rather long string containing
several lines of text just as you would do in C.

OUTPUT Note that whitespace at the beginning of the line is
significant.

ถ้าไม่ต้องการใช้สัญลักษณ์ \ ในการพิมพ์คำสั่งหลายบรรทัด ให้ใช้ """ (triple-quotes) แทนได้ เช่น

```
print """Usage: thingy [OPTIONS]
-h           Display this usage message
-H hostname   Hostname to connect to """)
```



Usage: thingy [OPTIONS]
-h Display this usage message
-H hostname Hostname to connect to

OUTPUT

สามารถเชื่อมต่อโดยการนำเอาข้อความสตริงมาต่อ กันได้ตรงๆ ภายใต้ " หรือ ' ได้ เช่น

```
str = 'Python' " is glue " 'language.'
print (str)
```



Python is glue language.

OUTPUT

สำหรับการคำนวณทางคณิตศาสตร์ สามารถสั่งพิมพ์ได้โดยตรง หรือผ่านตัวแปรก็ได้

```
print (5.0*2-25%4)
```

$$\text{ตัวอย่างสมการ } m_{stone} = \frac{mkg \times 22}{14} \text{ สมมติว่า mkg = 20}$$

```
mass_stone = mkg * 2.2 / 14
print("You weigh", mass_stone, "stone.")
```



9.0
You weigh 3.142857142857143 stone.

OUTPUT

ผู้ใช้สามารถพิมพ์ค่าของตัวแปรได้ โดยใช้สัญลักษณ์ , (Concatenation symbol) ด้านหลัง หรือ ด้านหน้าข้อความ ได้ เช่น $x = 8.5, y = 15$

```
x = 8.5; y = 15
```

```
print(x, "Test printing by using Concatenation ",y)
```



8.5 Test printing by using Concatenation 15

OUTPUT

การพิมพ์สตริง หรือสายอักขระ (String)

สายอักขระ (String) คือการเรียงต่อ กันของอักขระหลายๆ เข้าด้วยกัน เช่น ถ้าตัวอักขระ ประกอบด้วย 'H', 'e', 'l', 'l', 'o' (โดยปกติตัวอักขระจะเขียนอยู่ในเครื่องหมาย ' และสายอักขระ อยู่ภายใต้เครื่องหมาย " แต่ไฟรอนจะใช้เครื่องหมายทั้งสองแทนกันได้) เมื่อเรียงต่อเป็นสายอักขระจะได้ คำว่า "Hello" ผู้เขียนโปรแกรมสามารถควบคุมการทำงานของคำสั่ง print ให้มีลักษณะพิเศษเพิ่มขึ้น เช่น ขึ้นบรรทัดใหม่ การแท็บข้อความ (Tab) หรือ backspace เป็นต้น โดยใช้คำสั่ง Escape sequences ทำงานร่วมกับเครื่องหมาย ' หรือ " สำหรับรหัสคำสั่ง Escape แสดงในตารางที่ 3.1

ตารางที่ 3.1 Escaping sequences

Escape sequence	ความหมาย
\ (Backslash)	ไม่สนใจ (Ignored) หรือ ต้องการพิมพ์คำสั่งเดียวกัน 여러 ๆ
\\" (Double backslash)	พิมพ์เครื่องหมาย \ ออกจากภาพ
' (Single quote)	พิมพ์เครื่องหมาย ' ออกจากภาพ
\" (Double quote)	พิมพ์เครื่องหมาย " ออกจากภาพ
\n (Newline: LF)	ขึ้นบรรทัดใหม่ 1 บรรทัด
\r (Carriage return: CR)	เลื่อนเคอร์เซอร์ไปทางซ้ายมือสุดของบรรทัด

\t (Tab)	ตั้งแท็บในแนวนอน
\v (Vertical tab: VT)	ตั้งแท็บในแนวตั้ง
\e (Escape character: ESC)	คำสั่งให้ยกเลิกคำสั่งสุดท้าย
\f (Formfeed: FF)	ขีนหน้าใหม่
\b (Backspace)	เลื่อนเคอเซอร์ไปลบตัวอักษรทางซ้ายเมื่อหนึ่งตัวอักษร
\a (Bell: BEL)	สั่งให้เสียงกระดิ่งดังของลำโพงหนึ่งครั้ง

ตัวอย่างการใช้ Escape sequences

```
print("Example Heading\n\nFollowed by a line\nor two of text\n... \tName\n\tRace\n\tGender\n\nDon't forget to escape '\\\\'.")
```



Example Heading

OUTPUT

```
Followed by a line
or two of text
... Name
Race
Gender
Don't forget to escape '\\'.
```

จากตัวอย่าง หลังข้อความ Example Heading ใช้เครื่องหมาย \ ท่าก ซึ่งสั่งให้ขีนบรรทัดใหม่ 2 บรรทัด ก่อนจบบรรทัดแรกใช้เครื่องหมาย \ ซึ่งหมายถึงคำสั่งยังไม่จบ หรือหมดเพียงแค่ 1 บรรทัด เท่านั้น ยังมีคำสั่ง หรือข้อความต่อไปอีก ก่อนหน้าคำว่า Name ใช้เครื่องหมาย \t เพื่อย่อหน้า 1 แท็บ หลังข้อความ Don ใช้เครื่องหมาย \' เพื่อพิมพ์สัญลักษณ์ ' ร่วมกับข้อความ Don't และก่อนสิ้นสุดคำสั่ง จะพิมพ์สัญลักษณ์ ' โดยใช้กลุ่มของ Escape คือ \' และพิมพ์สัญลักษณ์ \' ด้วยกลุ่มสัญลักษณ์ \|

การแปลงรูปแบบสตริง (String formatting conversion)

ในบางกรณีการแสดงผลสตริง ไม่สะดวกที่จะใช้เครื่องหมาย +, " หรือ , ในการเชื่อมต่อ ดังนั้น ผู้เขียนโปรแกรมสามารถใช้เครื่องหมาย % (String formatting operator) ตามด้วยชนิดข้อมูล เช่น %s, %d (คล้าย printf ของภาษาซี) เป็นต้น เข้ามาช่วยในการนำค่าข้อมูลในตัวแปรมาแสดงผลร่วมกับคำสั่ง print ได้ ทำให้การแสดงผลข้อมูลมีความยืดหยุ่นขึ้น เช่น

```
Subject = "Python language"
print("I like to study the %s. " %Subject)
```



I like to study the Python language.

OUTPUT

จากตัวอย่างตัวแปร Subject เก็บข้อมูลชนิดสตริงคือ Python language ไว้ เมื่อใช้คำสั่ง print และตามด้วย %s (พิมพ์ข้อมูลที่เป็นสตริง) โปรแกรมจะรู้อัตโนมัติว่าจะนำค่าของข้อมูลในตัวแปร

Subject มาแสดงผลตรงตำแหน่ง %s ซึ่งเป็นการเชื่อมสตริงระหว่าง "I like to study the" กับ "Python language" เข้าด้วยกัน สำหรับสัญลักษณ์ที่ใช้ในการแปลงสายอักขระ หรือสตริง แสดงในตารางที่ 3.2

ตารางที่ 3.2 String formatting operator

Conversion Type	ความหมาย
%d, %i	เลขจำนวนเต็มแบบมีเครื่องหมาย เช่น 10, -5, 0
%u	เลขจำนวนเต็มแบบไม่มีเครื่องหมาย เช่น 0, 100, 1024
%o	เลขฐานแปดแบบไม่มีเครื่องหมาย เช่น 0, 1, 2, 3, 4, 5, 6, 7
%x	เลขฐานสิบหกแบบไม่มีเครื่องหมายและเป็นอักษรตัวเล็ก เช่น 1, a, c
%X	เลขฐานสิบหกแบบไม่มีเครื่องหมายและเป็นอักษรตัวใหญ่ เช่น 1, A, C
%e	เลขยกกำลัง และเป็นอักษรตัวเล็ก เช่น 1.000000e+06
%E	เลขยกกำลัง และเป็นอักษรตัวใหญ่ เช่น 1.000000E+06
%f, %F	เลขทศนิยม เช่น 3000.000000, -3.500000
%g	แสดงผลเหมือน e ถ้าข้อมูลมีค่าทศนิยมเกิน 4 หลัก เช่น 0.00001 จะให้ผลลัพธ์เป็น 1e-05 กรณีอื่นๆ จะแสดงผลเหมือน f เช่น 0.0001
%G	แสดงผลเหมือน E ถ้าข้อมูลมีค่าทศนิยมเกิน 4 หลัก เช่น 0.00001 จะให้ผลลัพธ์เป็น 1E-05 กรณีอื่นๆ จะแสดงผลเหมือน F เช่น 0.0001
%c	แสดงผลตัวอักษร ซึ่งใช้ได้กับ เลขจำนวนเต็ม หรือตัวอักษรเพียง 1 ตัว
%r	แสดงผลเป็นสตริง (แปลงจากอ้อปเจกต์ ไปเป็นสตริงโดยใช้ไลบรารี repr)
%s	แสดงผลเป็นสตริง (แปลงจากอ้อปเจกต์ ไปเป็นสตริงโดยใช้ไลบรารี str)
%%	แสดงผลเครื่องหมาย %

ตัวอย่างการใช้ String formatting operator

```

print ('Signed integer decimal: %d' %42)
print ('Unsigned octal: %o' %42)
print ('Unsigned decimal: %u' %42)
print ('Unsigned hexadecimal: %x, %X' %(15, 15))
print ('Floating-point exponential format: %f, %F' %(42, 100.435))

from math import pi    # import math library
print ('Floating-point decimal format: %f' %pi)
print ('Show g and G formating: %g %G' %(pi, 00.0000034))
print ('Single character: %c' %120)    # 120 = x in ASCII CODE
print ('Using str: %s' %42)
print ('Using repr: %r' %42)

```

OUTPUT



```

Signed integer decimal: 42
Unsigned octal: 52
Unsigned decimal: 42
Unsigned hexadecimal: f, F

```

```
Floating-point exponential format: 42.000000, 100.435000
Floating-point decimal format: 3.141593
Show g and G formating: 3.14159, 3.4E-06
Single character: x
Using str: 42
Using repr: 42
```

การแสดงผลเลขทศนิยม (Floating-point formatting)

การแสดงผลจำนวนทศนิยม โดยปกติจะมี 2 ส่วนคือ ด้านหน้า และด้านหลังจุดทศนิยม เช่น 100.35 ผู้เขียนโปรแกรมสามารถปรับขนาดความกว้างของจำนวนทศนิยมได้ จากตัวอย่าง 100 มีความกว้างเท่ากับ 3 และ .35 มีความกว้างเป็น 3 (รวมจุดทศนิยมด้วย) เป็นต้น เมื่อต้องการให้จำนวนทศนิยมมีขนาดความกว้างโดยรวมเป็น 10 ให้กำหนดดังนี้

```
print('%10f' % pi)

          +-----+ Width
          |       |
  10 9 8 7 6 5 4 3 2 1
  ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓
  3 . 1 4 1 5 9 3 3
  +-----+
OUTPUT      ' 3.141593 '
```

จะให้ผลลัพธ์ คือ ' 3.141593 ' สังเกตว่า ถ้าจำนวนตัวเลขไม่ถึงจำนวนที่ระบุไว้ โปรแกรมจะเติมช่องว่างให้อัตโนมัติ (ในตัวอย่างนี้เติมช่องว่างด้านหน้าเพิ่ม 2 ตำแหน่ง) ถ้าต้องการกำหนดความกว้างหลังจุดทศนิยม ให้กำหนดดังนี้ สมมติ ให้จำนวนตัวเลขหลังจุดทศนิยม เป็น 2 ตำแหน่ง คำสั่งที่ใช้คือ

```
print('%10.2f' % pi)

          +-----+
          |       |
  10 9 8 7 6 5 4 3 2 1
  ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓
  3 . 1 4 1 5 9 3 3
  +-----+
OUTPUT      ' 3.14 '
```

ผลลัพธ์ที่ได้คือ ' 3.14 ' และถ้าผู้ใช้ไม่สนใจความกว้างโดยรวม แต่สนใจเฉพาะจำนวนตัวเลขหลังจุดทศนิยมเท่านั้น ให้ผู้ใช้กำหนดดังนี้ print('%.2f' % pi) ผลลัพธ์ที่ได้รับคือ 3.14

ผู้เขียนโปรแกรมสามารถใช้คำสั่งการแสดงผลนี้กับสตริงก็ได้ ตัวอย่างเช่น str = 'Python Programming' เมื่อต้องการแสดงผลเฉพาะคำว่า Python ให้กำหนดดังนี้

```
str = 'Python Programming'
print('.6s' %str)

          +-----+
          |       |
  10 9 8 7 6 5 4 3 2 1
  ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓
  P y t h o n   P r o g r a m m i n g
  +-----+
OUTPUT      Python
```

ผู้เขียนโปรแกรมสามารถใช้สัญลักษณ์ * แทน ความกว้างโดยรวม และความกว้างหลังทศนิยมได้ โดยอ่านจากซุ่ดของตัวแปรที่ตามหลังมาได้ เช่น

```
print ('%.*s' % (5, 'Guido van Rossum'))
```

การเติมสัญลักษณ์ และจัดตำแหน่งหน้าทศนิยม

ผู้เขียนโปรแกรมสามารถกำหนดรูปแบบการแสดงผลด้านหน้าทศนิยมโดยการเติมเครื่องหมาย และจัดรูปแบบตำแหน่งการแสดงผลได้ดังนี้

```
from math import pi    # import pi from math library
print ('%010.2f' % pi)
```

 OUTPUT
0000003.14

จากตัวอย่าง ทำการแสดงผลค่า pi โดยกำหนดให้มีจำนวนตัวเลขหลังทศนิยมมีค่าเท่ากับ 2 ตำแหน่ง และความกว้างโดยรวมของการแสดงผลจำนวน 10 ตำแหน่ง ผลจากคำสั่งนี้ทำให้เกิดช่องว่างหน้าทศนิยม 6 ตำแหน่ง จึงทำการเพิ่ม 0 ไปด้านหน้าให้เต็ม 10 ตำแหน่ง เรียกวิธีนี้ว่าการแพดดิ้ง (Padding)



Note: การแพด (Padding) จะใช้ 0 (ศูนย์) ได้เพียงตัวเดียวเท่านั้น

ถ้าผู้เขียนโปรแกรมต้องการกำหนดค่าลบ (-) ให้กับจำนวนจริง จะไม่สามารถใส่เครื่องหมาย (-) เข้าไปได้ตรงๆ เพราะการใส่เครื่องหมายลบ ในชุดจะตีความว่าเป็นการจัดตำแหน่งให้กับจำนวนทศนิยมจากที่เคยแสดงผลชิดด้านขวา จะเปลี่ยนเป็นการแสดงผลทางด้านซ้ายแทน ตัวอย่างเช่น

```
from math import pi    # import pi
print ('%-10.2f' % pi)
```

 OUTPUT
'3.14' ,

ผลจากการรันคำสั่งดังกล่าว จะทำให้ค่าที่แสดงมาชิดทางด้านซ้ายแทน (การใช้เครื่องหมายลบหน้าทศนิยม คือการจัดตำแหน่งให้ชิดซ้าย) และด้านขวาจะมีค่าว่างจำนวน 6 ตำแหน่ง เมื่อต้องการกำหนดเครื่องหมาย ลบ หน้าทศนิยม ต้องกำหนดให้ตัวแปรมีค่าเป็นลบแทน ส่วนเครื่องหมายบวกสามารถกำหนดเข้าไปด้านหน้าทศนิยมได้ทันที เช่น

```
from math import pi
print ('%+10.2f' % pi)
print ('%10.2f' %-pi)
```



5. คำสั่งรับค่าข้อมูลจากแป้นพิมพ์ หรือคีย์บอร์ด (Keyboard Input)

การรับค่าข้อมูลจากแป้นพิมพ์ใน Python เวอร์ชันที่ต่ำกว่า 3.0 จะใช้ฟังชัน `input_raw()` สำหรับรับข้อมูลที่เป็นสายอักขระ หรือสตริง และฟังชัน `input()` สำหรับรับข้อมูลที่เป็นตัวเลข แต่ใน Python เวอร์ชัน 3.0 ขึ้นไป ได้ตัด `input_raw()` ทิ้งไป เหลือเพียง `input()` เท่านั้น แต่สามารถรับข้อมูลทั้งสองประเภทได้ เพื่อให้ผู้เขียนโปรแกรมสะดวก และยืดหยุ่นในการใช้งานมากขึ้น ซึ่งมีรูปแบบคือ

- การรับค่าข้อมูลที่เป็นสตริง หรืออ็อปเจกต์ :

```
<variable> = input("text")
```

```
เช่น s = input("What's your name : ") # รอรับการป้อนข้อมูลสตริงจากแป้นพิมพ์
print("Your name is :",s)
```

- การรับค่าข้อมูลที่เป็นตัวเลข หรือจำนวนจริง:

```
<variable> = int(input("text")) หรือ <variable> = float(input("text"))
```

```
เช่น age = int(input("How old are you : ")) # รอรับข้อมูลตัวเลขจำนวนเต็มจากแป้นพิมพ์
print("Your age is :",age)

VAT = float(input("Enter VAT : ")) # รอรับข้อมูลตัวเลขจำนวนจริงจากแป้นพิมพ์
print("Your VAT is :",VAT)
```

จากตัวอย่าง การรับข้อมูลจากแป้นพิมพ์โดยปกติจะเป็นสตริง (ถ้าไม่มีการทำ forcing หรือ casting) เมื่อผู้เขียนโปรแกรมต้องการใช้สำหรับคำนวน จำเป็นต้องแปลงจากสตริงเป็นจำนวนเต็ม หรือ จำนวนจริงเสียก่อน โดยทำ forcing เช่น `int(input("How old are you : "))` มีฉะนั้นจะเกิดข้อผิดพลาดในการคำนวนขึ้น

ทดสอบการทำงานของคำสั่ง `input()` ในการรับข้อมูลชนิดต่างๆ

```
>>> test = input("Enter float :")
Enter float :4.5      # User types input
>>> print (test)
4.5
```

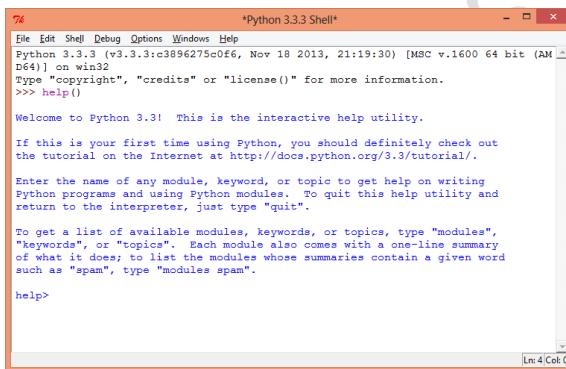
```
>>> type(test)
<class 'str'> # test is str not float
```

ทดสอบรับค่าอีกครั้งโดยทำการ forcing ให้เป็น float ก่อน

```
>>> test = float(input("Enter float :"))
Enter float :4.5 # User types float input
>>> type(test)
<class 'float'> # Float input was correct
```

6. คำสั่งช่วยเหลือ help()

ไฟรอนเตรียมคำสั่งไว้สำหรับช่วยเหลือผู้เขียนโปรแกรมในกรณีที่ไม่เข้าใจเกี่ยวกับการทำงานของคำสั่งต่างๆ โดยผู้ใช้งานสามารถใช้คำสั่ง help() เมื่อเรียกใช้งานคำสั่งดังกล่าวโปรแกรมจะเข้าสู่โหมดการช่วยเหลือดังรูปที่ 3.2



รูปที่ 3.2 การทำงานของไฟรอนในโหมดการช่วยเหลือ (help)

จากรูปผู้ใช้งานสามารถป้อนคำสั่งที่ไม่เข้าใจ หรือคำสั่งที่ต้องการเรียนรู้เข้าไปได้เลย ตัวอย่างเช่น ผู้ใช้ต้องการเรียนรู้คำสั่ง print ให้ป้อนคำสั่งคือ

Help> print จะได้ผลลัพธ์เป็น

 OUTPUT	<pre>Help on built-in function print in module builtins: print(*value, **kwargs) print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False) Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream. help></pre>
--	---

เมื่อต้องการออกจากโปรแกรม ให้ใช้คำสั่ง quit จะกลับเข้าสู่ Python shell

จบบทที่ 3

บทที่ 4

ตัวแปร การกำหนดค่า และชนิดข้อมูล (Variables, Assignment and Data Types)



ค่าคงที่ (Literal constants) คือ ข้อมูลที่เป็นค่าคงที่ ซึ่งค่าเหล่านี้ถือว่าเป็นค่าที่ไม่เปลี่ยนแปลง สำหรับการเขียนโปรแกรมแล้ว ค่าคงที่เหล่านี้จะถูกกำหนดให้กับตัวแปร เพื่อจุดประสงค์หลายๆ อย่าง ส่วนใหญ่จะสอดคล้องกับชนิดข้อมูล (Data type) เช่น constant = 5, VAT = 0.7 เป็นต้น

ตัวแปร (Variable) คือ ตัวแปร ที่ผู้เขียนโปรแกรมประกาศขึ้น สำหรับใช้เก็บข้อมูลที่ต้องการ เพื่อนำไปใช้ในการเขียนโปรแกรม เพื่อทำการประมวลผลข้อมูล เก็บข้อมูลในหน่วยความจำของที่ โปรแกรมทำงาน เช่น constant และ VAT เป็นต้น

1. หลักการตั้งชื่อตัวแปร (Identifier)

1. ตัวอักษรตัวแรกต้องเป็นภาษาอังกฤษ (A-Z, a-z) ตามด้วยตัวอักษร หรือตัวเลขใดๆ ก็ได้ เช่น Score, SCORE1, s5, Test_para_01 เป็นต้น
2. ชื่อตัวแปรห้ามมีช่องว่าง จุดทศนิยม และสัญลักษณ์พิเศษ ยกเว้น underscore "_" เท่านั้น เช่น Count_01, str_, _doc, ____main, __func____, oo_x_oo เป็นต้น
3. การใช้อักษรตัวพิมพ์ใหญ่ และอักษรพิมพ์เล็กมีความแตกต่างกัน (Case-sensitive) เช่น Var1 กับ var1 ถือว่าไม่ใช่ตัวแปรเดียวกัน
4. ห้ามใช้คำสlang เป็นชื่อตัวแปร (Reserved word, Keyword) เช่น if, for, max, sum เป็นต้น
5. ควรจะตั้งชื่อให้สื่อกับความหมายใกล้เคียงกับค่าที่จะเก็บ สามารถอ่าน และทำความเข้าใจ ได้ง่าย เช่น Count สำหรับเก็บจำนวนหนับ Salary สำหรับเก็บเงินเดือน และ Total เก็บค่า ผลรวม เป็นต้น
6. ห้ามใช้เครื่องหมายต่อไปนี้ในการตั้งชื่อตัวแปร !,@, #, \$, %, ^, &, *, (,), -, =, \, |, +, ~, .

7. ตัวแปรไม่ควรยาวเกิน 255 ตัวอักษร ตัวแปรที่มีความยาวมากๆ หรือเป็นการผสมระหว่างคำ ให้ใช้ "_" เชื่อมคำเหล่านั้นแทน เช่น Thai_Market_Chair

2. การใช้งานตัวแปร (Variables using)

การใช้งานตัวแปรมี 3 ขั้นตอน คือ

- การประกาศตัวแปร (Variable declaration) ก่อนใช้งานตัวแปรใดๆ จำเป็นต้องประกาศให้คอมไพล์รู้เสียก่อน ตามหลักการแล้วจะต้องประกาศค่าตัวแปรให้สอดคล้องกับข้อมูลที่จะนำไปใช้ เช่น `int x = 5` แต่ไฟรอนไม่ได้ให้ความสำคัญกับการประกาศชนิดของตัวแปร ทำให้ผู้เขียนโปรแกรมไม่จำเป็นต้องกังวลว่าควรเลือกใช้ตัวแปรชนิดใดให้เหมาะสมกับงาน การแยกแยะชนิดของตัวแปรจะเป็นหน้าที่ของไฟรอน ซึ่งจะทำให้เองแบบอัตโนมัติ โดยพิจารณาจากค่าข้อมูลที่กำหนดให้กับตัวแปรนั้นๆ เช่น

```
Price = 120          #ไฟรอนจะพิจารณาว่าเป็นจำนวนเต็ม
VAT = 0.07           #จำนวนจริง
Display = "Calculating the price goods" #สตริง
Total = Price + VAT #ไฟรอนตีความว่า Total เป็นจำนวนจริง
```

- กำหนดค่าให้ตัวแปร (Assigning values to variables) มีรูปแบบดังนี้ คือ
ชื่อตัวแปร = ค่าของข้อมูล เช่น

```
Name = "Suchart"      #กำหนดสตริงให้กับตัวแปร Name
String = ""            #กำหนดค่าว่างให้กับตัวแปร String
TAX = 0.075           #กำหนดค่าจำนวนจริงให้กับตัวแปร TAX
```

ด้านซ้ายมือของเครื่องหมาย = เป็นชื่อตัวแปร ส่วนด้านขวา มือ คือข้อมูลที่ต้องการกำหนดลงไปโดยปกติแล้ว เมื่อทำการประกาศตัวแปรในไฟรอนจะต้องมีการกำหนดค่าเริ่มต้นให้ตัวแปรเสมอ (มิเช่นนั้นจะเกิดข้อผิดพลาด) แต่ถ้าผู้เขียนโปรแกรมยังไม่แน่ใจว่าควรจะกำหนดค่าอะไร หรือเตรียมตัวแปรไว้ร่วงหน้า เพื่อรอค่าที่จะเก็บในภายหลัง ให้กำหนดด้วย "" หรือ " สำหรับสตริง หรือ None (ค่าว่าง หรือ NULL) ถ้าเป็นจำนวนเต็มควรเป็น 0 จำนวนจริงเป็น 0.0 เป็นต้น สังเกตการกำหนดค่าให้ตัวแปร ดังต่อไปนี้

```
counter = 100          # Integer
miles    = 1000.0       # Floating
name     = "John"        # String
print (counter)
```

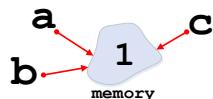
```
print (miles)
print (name)
```

 100
1000.0
OUTPUT John

การกำหนดค่าตัวแปรหลาย ๆ ค่าพร้อมกัน

ไฟชอนอนุญาตให้ผู้ใช้สามารถกำหนดค่าตัวแปรได้หลายค่าพร้อมๆ กัน เช่น

a = b = c = 1



ค่าในตัวแปร a, b และ c มีค่าเท่ากับ 1

a, b, c = 1, 2, "john"



ค่าในตัวแปร a มีค่าเท่ากับ 1, ตัวแปร b เท่ากับ 2 และ c มีค่าเท่ากับ “John” หรือผู้ใช้สามารถแทนค่าตัวแปรได้โดยลักษณะนิพจน์ (Expression) เช่น

```
x1, y1 = 2, 3           # x1 = 2, y1 = 3
x2, y2 = 6, 8           # x2 = 6, y2 = 8
m, b = float(y1-y2)/(x1-x2), y1-float(y1-y2)/(x1-x2)*x1
print ("y=",m,", b =",b)
```

 y= 1.25 , b = 0.5
OUTPUT

ผู้ใช้สามารถกำหนดค่าตัวแปรหลาย ๆ ค่า คล้ายภาษาซีได้ โดยใช้เครื่องหมาย ; คันแต่ละตัวแปร ซึ่งมีรูปแบบ คือ

x = 8.5; y = 15; z = "jack"

ตัวแปร x จะมีค่าเท่ากับ 8.5, y = 15 และ z มีค่าเท่ากับ jack ตามลำดับ

3. การใช้ตัวแปร (Use the variable) ตัวแปรต้องมีการประกาศ และกำหนดค่าไว้แล้วเท่านั้น จึงจะสามารถนำมาใช้งานได้ ในการทำงานจริง ๆ นั้น ตัวแปรเป็นค่าที่สามารถเปลี่ยนแปลงได้ขึ้นอยู่กับเงื่อนไขการใช้งาน ซึ่งข้อมูลในตัวแปรส่วนใหญ่จะมีค่าคงที่ที่ไม่เคยเปลี่ยน เช่น ภาษีมูลค่าเพิ่มปัจจุบันเท่ากับ 7% ดังนั้น ตัวแปร VAT จะเท่ากับ 0.07 เสมอ ยกเว้นว่ารัฐบาลจะมีนโยบายเปลี่ยนแปลงภาษีใหม่ โปรแกรมเมอร์จึงจะแก้ไขค่าดังกล่าวในภายหลัง



Note: ไฟรอนมีคุณสมบัติการคืนหน่วยความจำให้ระบบ (Garbage collection) แต่ผู้ใช้สามารถคืนหน่วยความจำจากการประกาศตัวแปรได้ โดยใช้คำสั่ง `del` ชื่อตัวแปร เช่น `del a`



Tips: การกำหนดตัวแปรว่างเปล่า โดยใช้ "" จะทำให้ไฟรอนมองว่าเป็นข้อมูลชนิดสตริง แต่ถ้าผู้เขียนโปรแกรมไม่ต้องการให้ไฟรอนตีความว่าเป็นข้อมูลชนิดใดๆ ให้ใช้ `None` (ขึ้นต้นด้วย `N` ตัวใหญ่) เช่น `Var1 = None`

3. คำส่วน (Reserved word, Keyword)

คำส่วน คือคำที่ถูกภาษาไฟรอนใช้เพื่อสร้างไวยกรณ์ ดังนั้นผู้เขียนโปรแกรมห้ามนำไปใช้ในการสร้าง หรือประกาศเป็นตัวแปรโดยเด็ดขาด เพราะจะทำให้เกิดข้อผิดพลาด คือ `SyntaxError: invalid syntax`) เช่น ประกาศตัวแปร `if = 5` เป็นต้น สำหรับคำส่วนในภาษาไฟรอน ดังต่อไปนี้คือ

`and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, yield`

คำต่อไปนี้แม้ว่าไฟรอนไม่ได้ห้ามไว้แต่ก็ไม่ควรใช้ เพราะไปตรงกับชื่อของฟังชันในไฟรอน คือ `data, float, Int, numeric, Oxphys, array, close, int, input, open, range, type, write, zeros`

คำต่อไปนี้ก็ควรหลีกเลี่ยงด้วย ถ้ามีการนำเข้า (`import`) ไลบรารี `math` มาใช้งาน คือ `acos, asin, atan, cos, e, exp, fabs, floor, log, log10, pi, sin, sqrt, tan`



Tips: ถ้าผู้เขียนโปรแกรมจำเป็นต้องการตั้งชื่อให้เหมือนคำส่วนจริงๆ สามารถทำได้โดยใช้ `__`, อักษรตัวใหญ่ หรือเติมอักษรเพิ่ม เช่น `__print, Print, PRINT, print_msg` เป็นต้น

4. ชนิดข้อมูล (Data types)

ชนิดข้อมูลในภาษาไฟรอนแบ่งออกเป็น 2 กลุ่มหลักๆ คือ ข้อมูลพื้นฐาน (Basic data types) และข้อมูลเชิงประกอบ (Composite data types)

1. **ข้อมูลพื้นฐาน (Basic data types)** แบ่งออกเป็น 2 กลุ่มย่อย คือ ข้อมูลที่เกี่ยวข้องกับตัวเลข (`Numeric`) และข้อมูลสายอักษร (`String`)

1) ข้อมูลตัวเลข (Numeric)

ข้อมูลตัวเลข หมายถึงชนิดข้อมูลที่สามารถเก็บข้อมูลที่เป็นข้อมูลทั่วไป หรือข้อมูลพื้นฐาน เช่น เลขจำนวนนับ ซึ่งเลขจำนวนนับนี้มีคุณสมบัติสามารถเพิ่มค่าได้ คำนวนได้ และเปลี่ยนแปลงค่าได้ มีทั้งหมด 4 ชนิด ได้แก่ เลขจำนวนเต็ม (Integers) ตัวเลขทศนิยม หรือจำนวนจริง (Floating-Point numbers) จำนวนตรรกะ (Boolean) และจำนวนเชิงซ้อน (Complex Numbers)

● เลขจำนวนเต็ม (Integers)

เลขจำนวนเต็มในไพธอนเวอร์ชันก่อน 3.0 แบ่งออกเป็น 2 ประเภทคือ จำนวนเต็มธรรมดา (Plain integers) และจำนวนเต็มแบบยาว (Long integers) แต่สำหรับ Python 3.0 ขึ้นไป ถูกออกแบบใหม่ให้เหลือแค่เลขจำนวนเต็มธรรมดาที่สามารถเก็บความยาวได้ไม่จำกัด ขึ้นอยู่กับจำนวนของหน่วยความจำที่เครื่องมีอยู่ (เก็บในหน่วยความจำแทนเก็บในรีจิสเตอร์) ถึงแม้ว่าการทำางจะซักกว่าแบบเดิม แต่ให้ความสะดวกในการเขียนโปรแกรมกับจำนวนเต็มที่มีขนาดใหญ่มากๆ ได้ดีกว่า เลขจำนวนเต็มสามารถแสดงได้หลายแบบคือ จำนวนเต็มแบบฐานสิบ เช่น 234, 0, -34 แบบฐานสอง (Binary) เช่น 0b1000111000 แบบฐานแปด (Octal) เช่น 0o746320 แบบฐานสิบหก (Hexadecimal) เช่น 0xDECADE และตรรกะ (บูลีน) เช่น 0, 1 หรือ True, False เป็นต้น ดังแสดงในตัวอย่าง

```
>>> 14600926 # decimal  
14600926  
  
>>> 0b1101110100101011011110 # binary  
14600926  
  
>>> 0o67545336 # octal  
14600926  
  
>>> 0xDECADE # hexadecimal  
14600926
```

การแสดงผลของเลขฐานสอง แปด และสิบหก จะถูกแสดงเป็นเลขฐานสิบแทน เพราะง่ายต่อการตีความหมาย เนื่องจากมนุษย์ใช้เลขฐานสิบในการทำงาน สำหรับเลขฐานสิบ ไม่ต้องเขียนเลขศูนย์นำหน้า เช่น 020 เพราะจะทำให้เกิดข้อผิดพลาดขึ้น เลขฐานสองจะต้องมี 0b (ศูนย์และบี) นำหน้า เลขฐานแปดต้องมี 0o (ศูนย์และโอ) เลขฐานสิบหกต้องมี 0x นำหน้า ตามลำดับ

เลขฐานแปด ประกอบไปด้วยตัวเลข ตั้งแต่ 0 – 7 เท่านั้น เช่น 0o123 มีค่าเท่ากับ 83 ในเลขฐานสิบ สำหรับเลขฐานสิบหกประกอบไปด้วยตัวเลข 0 – 9 และ A – F (A มีค่าเท่ากับ 10 ในเลขฐานสิบ B มีค่าเท่ากับ 11 และ F มีค่าเท่ากับ 15 ตามลำดับ) เช่น 0x2BC8 มีค่าเท่ากับ 11,208 ในเลขฐานสิบ สำหรับการแปลงค่าเลขฐาน จากฐานสอง แปด และสิบหกเป็นฐานสิบ จะอยู่นอกขอบเขตของหนังสือเล่มนี้ ผู้อ่านสามารถอ่านเพิ่มเติมได้จากหนังสือพื้นฐานคอมพิวเตอร์ทั่วไป

ผู้เขียนโปรแกรมสามารถทดสอบชนิดของตัวแปรด้วยคำสั่ง type(ตัวแปร) เช่น

```
>>> type(123)
<class 'int'>
>>> type(0b111100011)
<class 'int'>
>>> type(0o145)
<class 'int'>
>>> type(0xAB3)
<class 'int'>
```

จากตัวอย่างเลขฐานสิบ ฐานสอง ฐานแปด และฐานสิบหก เป็นตัวแปรชนิด integer (ใช้ตัวย่อ int)

ผู้เขียนโปรแกรมสามารถสั่งดำเนินการทางด้านคณิตศาสตร์ได้ๆ กับเลขจำนวนเต็มผ่านทาง Python shell ได้ทันทีเช่น

```
>>> 255 + 100
355
>>> 397 - 42
355
>>> 71 * 5
355
>>> 355 / 113
3
>>> 400 + 5; 300 - 4; -35 / 6 #combination commands
405
296
-5.833333333333333
```

● ตัวเลขทศนิยม หรือจำนวนจริง (Floating-Point numbers)

เลขจำนวนจริง หรือเรียกว่า float คือจำนวนที่มีทศนิยม ซึ่งสามารถเขียนได้ 2 รูปแบบ คือ เขียนตัวเลขทศนิยมที่มีเครื่องหมายจุดทศนิยม ตัวอย่างเช่น 3.14 หรือ เขียนอยู่ในรูปเลขยกกำลังสิบ (Exponential form) โดยใช้ตัวอักษร E หรือ e ระบุจำนวนที่เป็นเลขยกกำลัง เช่น $6.12E3 = 6.12 \times 10^3$ หรือ $125.03E-5 = 125.03 \times 10^{-5}$ เป็นต้น โดยภาษาโปรแกรมมิ่งทั่วๆไป จำนวนจริงจะมีสองแบบคือ float และ double โดย float ใช้สำหรับเก็บจำนวนที่มีความเที่ยงตรงตามปกติ ส่วน double ใช้สำหรับจำนวนที่ต้องการความเที่ยงตรงเป็นสองเท่า ความเที่ยงตรง (precision) ในที่นี้หมายถึงจำนวนหลักของตัวเลขหลังจุดทศนิยม ถ้ามีหลายหลัก ตัวเลขก็จะยิ่งถูกต้องเที่ยงตรงมากขึ้น สำหรับ Python จะไม่มีตัวแปรชนิด double มีเพียงเฉพาะ float แต่มีความสามารถเท่ากับ double สามารถใช้แทนกันได้ โดยมีช่วงค่าข้อมูลตั้งแต่ $2.2250738585072014e-308$ จนถึง $1.7976931348623157e+308$ ทั้งนี้ขึ้นอยู่กับสถาปัตยกรรมของเครื่องตัว� ผู้เขียนโปรแกรมสามารถใช้คำสั่งด้านล่างเพื่อทดสอบขนาดของ float ดังนี้

```
>>> import sys          # import library sys
>>> sys.float_info    # call function float.info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
```

```
min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

ตัวอย่างการใช้งานตัวแปร float

```
>>> 6/8
0.75
>>> 1./5.
0.2
>>> 2.3/25.7
0.08949416342412451
>>> type(3.5)
<class 'float'>
```



Tips: การบวก ลบ คูณ และหาร ระหว่าง Integer กับ Float จะให้ผลลัพธ์เป็น Float เสมอ ถ้าต้องการให้ผลลัพธ์เป็น Integer จะต้องกระทำการ casting ด้วยกันเองเท่านั้น หรือทำการ casting แต่จะทำให้ข้อมูลอาจจะผิดพลาดได้ ถ้าขนาดของ Float ใหญ่กว่า Integer

- จำนวนตรรกะ (Boolean)

ตัวแปรตรรกะ ใช้คำย่อในการเขียนโปรแกรมคือ bool เป็นชนิดของตัวแปรที่สามารถเก็บค่า ลอจิก จริง (True) หรือ เท็จ (False) ตัวแปรชนิดนี้ เป็นที่รู้จักกันอีกชื่อคือ ตัวแปรบูลีน (Boolean) ตัวอย่างค่าที่ถูกเก็บในตัวแปรชนิด bool "ได้แก่ 1 = True และ 0 = False"

ตัวอย่างการใช้งาน bool ดังนี้

```
>>> t = True          # t = 1
>>> f = False         # f = 0
>>> t and f          # 1 and 0 = 0
False
>>> t and True        # 1 and True = 1
True
>>> type(True)
<class 'bool'>
```

- จำนวนเชิงซ้อน (Complex numbers)

จำนวนเชิงซ้อนคือ ตัวเลขสองมิติประกอบไปด้วย มิติของเลขจริง และมิติของเลขจินตภาพ เขียนอยู่ในรูป $z = x + yi$ เมื่อ x เป็นตัวเลขในแกนจริง (Real axis) และ y เป็นเลขในแกนจินตภาพ (Imaginary axis) เช่น จำนวนเชิงซ้อน $2 + 7i$ มีค่าแกนจริงเป็น 2 และค่าแกนจินตภาพเป็น 7 ตัว i นั้น อาจจะมองว่าเป็นตัวกำกับแกน คือแสดงให้รู้ว่า ตัวเลขที่ติดกันนั้นหมายถึงเลขในแกนจินตภาพ ในทางคณิตศาสตร์นั้น ตัว i จะมีคุณสมบัติต่างๆ เมื่อนำตัวเลขโดยที่ i^2 มีค่าเท่ากับ -1

ตัวอย่างการใช้จำนวนเชิงซ้อน

```

>>> real = 8                  # real axis
>>> imag = 4j                 # imaginary axis
>>> z = real + imag          # Construct complex number
>>> z                         # Display complex number
(8+4j)
>>> z = -89.5 + 2.125j
>>> z.real, z.imag           # z.real = real axis
(-89.5, 2.125)
>>> type(z)
<class 'complex'>

>>> z = complex(2,3)          # Function constructs complex
>>> z
(2+3j)
>>> x = 3 + 4j                # To do operations on complex
>>> y = 2 - 3j
>>> z = x + y
>>> print (z)
(5+1j)

```

จากตัวอย่าง real เก็บค่าของตัวเลขของแกนจริง และ imag เป็นค่าของแกนจินตภาพ สำหรับตัวแปลล์ z จะเก็บค่าของจำนวนเชิงซ้อนที่เกิดจาก การรวมกันของแกนจริง และแกนจินตภาพ ถ้าต้องการอ้างแกนใดแกนหนึ่งสามารถทำได้โดยการอ้างอิงไปยังตัวอักษรตัวอักษรของจำนวนเชิงซ้อน แล้วตามด้วย “.” ต่อด้วยชื่อพังชัน เช่น z.real = อ้างอิงแกนจริง, z.imag = อ้างอิงแกนจินตภาพ ในกรณีสุดท้ายผู้เขียนโปรแกรมสามารถสร้างจำนวนเชิงซ้อนได้ด้วยเรียกผ่านพังชัน complex (real, imag) ผลลัพธ์ที่ได้จะเป็นจำนวนเชิงซ้อนเหมือนตัวอย่างข้างต้น ถ้าผู้เขียนโปรแกรมต้องการเปลี่ยนเครื่องหมายของค่าในแกนจินตภาพ (Conjugate) สามารถทำได้โดยการเรียกพังชัน conjugate() เช่น

```

>>> z.conjugate()
(8-4j)
>>> (3 -5j).conjugate()
(3+5j)

```



Note: คณิตศาสตร์พื้นฐานจะใช้ i แทน $\sqrt{-1}$ แต่ในไฟร์ฟอนเลือกใช้ j แทน เพราะเป็นตัวแปรที่นิยมในงานวิศวกรรมศาสตร์

- การเปลี่ยนค่าตัวแปร (Forcing a number type)

ผู้เขียนโปรแกรมสามารถแปลงค่าไปมาระหว่างตัวแปรต่างชนิดกันได้ โดยใช้การ forcing ซึ่งสามารถทำได้ดังนี้

การแปลงจากสตริงเป็นจำนวนเต็ม (String to Integer)

```

>>> x = int("17")
>>> y = int(4.8)
>>> print ("x =",x, ",y =",y, " and x - y =",x - y)
x = 17 ,y = 4 and x - y = 13

```

การแปลงจากสตริงเป็นจำนวนจริง (String to Float)

```
>>> x = float(17)
>>> y = float("4")
>>> print ("x =", x, ", y =", y, " and x - y =", x - y)
x = 17.0 ,y = 4.0 and x - y = 13.0
```



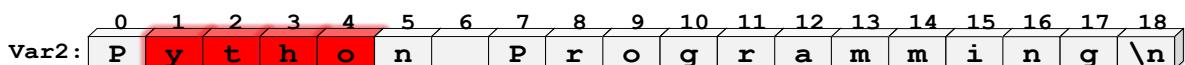
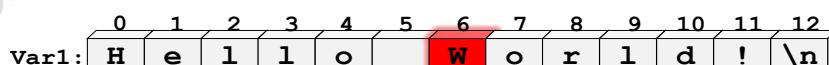
Tips: ถ้าผู้เขียนโปรแกรมต้องการทราบชนิดของตัวแปร ให้ใช้คำสั่ง type (ตัวแปร) เช่น type ("Hello") ผลลัพธ์คือ <class 'str'>, type (17) ผลลัพธ์คือ <class 'int'>, type(1.0) = <class 'float'> ตามลำดับ

2) ข้อมูลชนิดสายอักขระ (String)

ข้อมูลสายอักขระ หรือสตริง หมายถึง ข้อมูลที่เป็น ตัวอักขระ ข้อความ หรือประโยค ซึ่งตัวแปรชนิดนี้ไม่สามารถนำมาคำนวณได้ ในการประมวลผลตัวแปรชนิดนี้ ข้อความจะต้องอยู่ภายใต้เครื่องหมาย (" ") หรือเครื่องหมาย (' ') กำกับอยู่ เช่น author = 'Suchart' หรือ author = "Suchart" ดังนั้นในกรณีที่มีการเก็บในลักษณะเป็นตัวเลข เช่น '15.25' จึงมีความหมายเป็นเพียงสายอักขระ ไม่สามารถนำมาประมวลผลได้ แต่ถ้าผู้เขียนโปรแกรมต้องการให้คำนวณได้ จำเป็นต้องใช้ฟังก์ชันเพื่อเปลี่ยนชนิดตัวแปร (Forcing) จากสายอักขระไปเป็นจำนวนเต็ม หรือจำนวนจริง จึงจะสามารถประมวลผลได้ วิธีการเขียนคำสั่งเพื่อเก็บข้อมูลชนิดตัวแปรสายอักขระ ดังนี้

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

ประกาศให้ตัวแปร var1 มีค่าเท่ากับ Hello World! และ var2 เท่ากับ Python Programming ภาษาไฟรอนไม่สนับสนุนตัวแปรแบบอักขระ เช่น 'A', '1' ดังนั้น ถ้าต้องการใช้งานลักษณะดังกล่าวจะต้องทำการระบุลำดับของตัวอักษรนั้นๆ ในสตริงแทน โดยใช้เครื่องหมาย [] เข้าช่วย เช่น ถ้าต้องการข้อมูล 'W' ในตัวแปร var1 ทำได้โดยอ้างตัวแปรและตามด้วย [ตำแหน่งของตัวอักษร] เช่น var1[6] เป็นต้น ดังตัวอย่าง



```
>>> print ("var1[6]: ", var1[6])
>>> print ("var2[1:5]: ", var2[1:5])
var1[6]: w
var2[1:5]: ytho
```

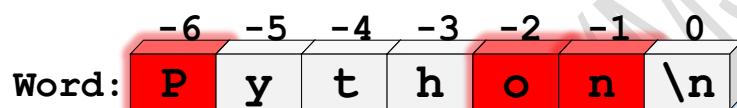
จากตัวอย่าง ผู้เขียนโปรแกรมสามารถเข้าถึงข้อมูลเฉพาะบางส่วนของสตริงได้ โดยใช้คำสั่ง

ชื่อตัวแปร[ตำแหน่งเริ่มต้น : ตำแหน่งสิ้นสุด] เช่น var2[1:5] ค่าที่ได้คือ ytho



Note: ควรจำไว้ว่า คำสั่ง [1:5] จะได้ข้อมูลตำแหน่งที่ 1 – 4 เท่านั้น ไม่รวมตำแหน่งที่ 5

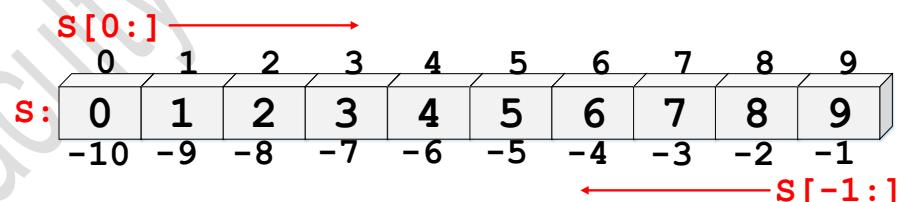
การเข้าถึงข้อมูลของสตริง โดยปกติจะเข้าถึงจากส่วนหัวของสตริง หรือเข้าถึงแบบสุ่มโดยอาศัยเครื่องหมาย [] แต่ในเมื่อการเข้าถึงข้อมูลจากส่วนท้ายของสตริงให้ด้วย (ปกติความยาวของสตริงจะไม่แน่นอน ดังนั้นการเข้าถึงข้อมูลจากส่วนท้ายของสตริงจะทำได้ยากกว่าวิธีการเข้าถึงแบบปกติ) โดยอาศัยเครื่องหมาย [] ร่วมกับจำนวนเต็มลบ เช่น ถ้าต้องการเข้าถึงข้อมูลตัวสุดท้ายของสตริงจะใช้ [-1] ตัวที่สองจากส่วนท้ายของสตริงคือ [-2] ตามลำดับ ดังตัวอย่าง



```
>>> word = 'Python'
>>> word[-1]           # last character
'n'
>>> word[-2]           # second-last character
'o'
>>> word[-6]           # last-last character
'P'
```

การเข้าถึงข้อมูลของสตริงบางส่วน เช่น บอกเฉพาะตำแหน่งเริ่มต้นเป็นต้นไป หรือบอกเฉพาะตำแหน่งสิ้นสุด หรือบอกตำแหน่งการกระโดด สามารถทำได้โดยใช้รูปแบบคำสั่งคือ

สมมุติว่า s = '0123456789'

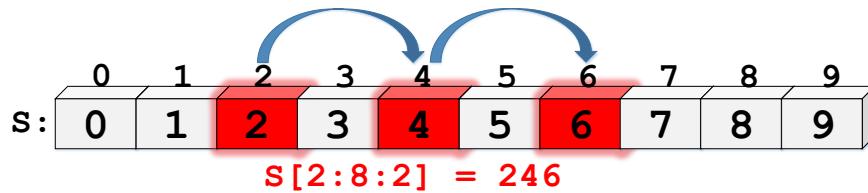


1. s[start] บอกเฉพาะตำแหน่งเริ่มต้นของข้อมูลเท่านั้น

```
print(s[2]) == '2'
```
2. s[start:end] บอกตำแหน่งเริ่มต้น และสิ้นสุดข้อมูล

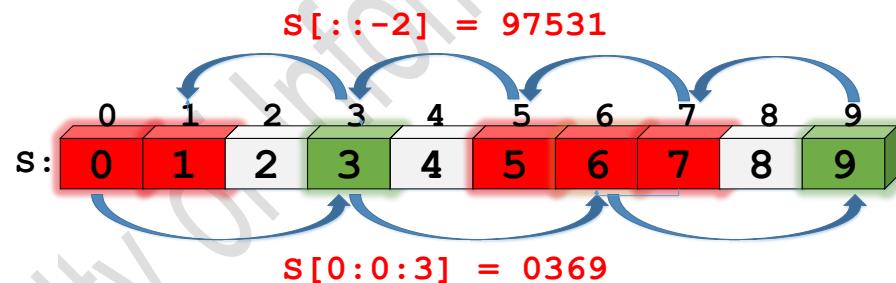
```
print(s[3:6]) == '345'
```
3. s[start:end:step] บอกตำแหน่งเริ่มต้น สิ้นสุด และลำดับการกระโดด ในตัวอย่างจะเริ่มจาก 2 และทำการกระโดดไปครั้งละ 2 ตำแหน่ง ไปสิ้นสุดที่ 8

```
print(s[2:8:2]) == '246'
```



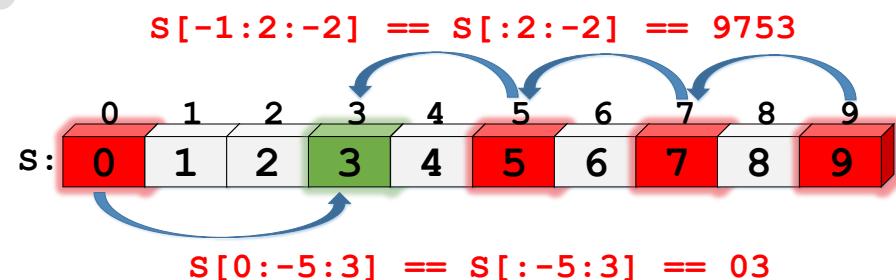
ในกรณีนี้จะทำการกระโดดไปครั้งละ 2 ตำแหน่ง โดยเริ่มจาก 2, 4 และ 6 ตามลำดับ

4. $s[:end]$ ไม่บอกตำแหน่งเริ่มต้น บอกเฉพาะตำแหน่งสิ้นสุด ถ้าไม่บอกตำแหน่งเริ่มต้น โปรแกรมจะเริ่มต้นที่ตำแหน่ง 0 – end เช่น
`print(s[:5]) == '01234'`
5. $s[start:]$ ไม่บอกตำแหน่งสิ้นสุด บอกเฉพาะตำแหน่งเริ่มต้น ถ้าไม่บอกตำแหน่งสิ้นสุด โปรแกรมจะเริ่มต้นที่ตำแหน่ง start – ท้ายสุดของสตริง เช่น
`print(s[4:]) == '456789'`
6. $[:3]$ บอกเฉพาะจำนวนตำแหน่งที่กระโดดไปเท่านั้น จนครบทั้งสตริง ในตัวอย่างจะกระโดดทีละ 3 ตำแหน่ง เช่น
`print(s[::-3]) == '0369'`
7. $[:-2]$ บอกเฉพาะจำนวนตำแหน่งที่กระโดด และแสดงผลข้อมูลแบบถอยหลัง ทีละ 2 ตำแหน่ง เช่น
`print(s[::-2]) == '97531'`



8. $[-1:2:-2]$ เริ่มต้นจากท้ายสตริง จนถึงตำแหน่งที่ 2 จากด้านหัวสตริง และกระโดดครั้งละ 2 ตำแหน่ง เช่น

```
print(s[-1:2:-2]) = '9753'
```



$s[0:-5:3] == s[-5:3] == 03$

การเพิ่ม หรือแก้ไขข้อมูลสตริง สามารถทำได้ทันที โดยการกำหนดค่าใหม่ให้กับตัวแปรที่ต้องการแก้ไข

```
>>> var1 = 'Hello World!'
print ("Updated String :- ", var1[:6] + 'Python
language')
Updated String :- Hello Python language
>>> var1 = 'This is the new string.'
print ("Replaced String :- ", var1)
Replaced String :- This is the new string.
```

จากตัวอย่าง คำสั่ง var1[:6] คือการแก้ไขค่าข้อมูลใน var1 ตั้งแต่ตำแหน่งข้อมูลตัวที่ 6 เป็นต้นไป จากเดิมคือ World! ไปเป็น Python language และตัวอย่างถัดมาเป็นการแทนค่าสตริงใหม่ทั้งหมด



Note: ควรจำไว้ว่า ผู้เขียนโปรแกรมไม่สามารถแก้ไขสตริงโดยการระบุตำแหน่งได้โดยตรง เช่น s[0] = 'C' ต้องใช้เมธอด s.replace() แทน เพราะสตริงเป็นตัวแปรแบบ Immutable ในบางกรณีที่ผู้ใช้ต้องการพิมพ์ค่าข้อมูล หรือผลลัพธ์ ไปพร้อมๆ กับการพิมพ์ข้อความ สามารถทำได้โดยใช้ , ช่วยในการพิมพ์ เช่น

```
>>> i = 10
>>> print ("15 / 3 = ", 15 / 3)
15 / 3 = 4.666666666666667
>>> print (15 % 3, " = 15 % 3")
0 = 15 % 3
>>> print ("15.0 / 3.0 =", 15.0 / 3.0, " Baht")
15.0 / 3.0 = 5.0 Baht
>>> print ("This is ", "the integer =", i)
This is the integer = 10
```

การเชื่อมต่อสตริงสามารถทำได้โดยใช้เครื่องหมาย + (String concatenation symbol) ดังนี้

```
>>>Str1 = "Python is a widely used general-purpose,"
>>>Str2 = 'high-level programming language.'
>>>Str3 = 'Its design philosophy emphasizes code readability, \
and its syntax allows programmers to express concepts in \
fewer lines of code than would be possible in languages such \
as C'
>>>Python = Str1 + Str2 + Str3 # String concatenation
>>>print(Python)
Python is a widely used general-purpose,high-level programming
language.Its design philosophy emphasizes code readability, and its
syntax allows programmers to express concepts in fewer lines of
code than would be possible in languages such as C
```

ผู้เขียนโปรแกรมสามารถใช้เครื่องหมาย ' และ " แทนกันได้ แต่มีหลักการใช้คือ ถ้ากลุ่มของข้อความจำเป็นต้องมีเครื่องหมาย ' อุปภาษาในข้อความ ควรใช้เครื่องหมาย " ครอบอุปภาษานอกสายอักษร ในทางกลับกัน ถ้าในข้อความมีเครื่องหมาย " อุปภาษาใน ควรจะใช้ ' ครอบสายอักษรไว้ หรือสามารถใช้ \ เพื่อบอกกับไฟชอนว่าไม่ต้องเปลี่ยนความหมายของอักษร (เพียง 1 ตัวอักษรเท่านั้น) ที่ตามหลังเครื่องหมายดังกล่าวมาก็ได้ ดังตัวอย่างด้านล่าง

```

>>> 'spam eggs' #single quotes
'spam eggs'
>>> 'doesn\'t' # use \' to escape the single quote...
"doesn't"
>>> "doesn't" # ...or use double quotes instead
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'

```

เมื่อผู้ใช้พิมพ์คำสั่งเข้าไปโดยตรงที่ Python shell หรือ Interactive interpreter จะปรากฏรหัสควบคุม (') ที่อยู่ภายนอกข้อความติดมาด้วย แต่เมื่อพิมพ์ผ่านฟังชัน print จะทำให้เครื่องหมายที่ครอบข้อความอยู่หายไป เช่น

```

>>> '"Isn\'t," she said.'
'"Isn\'t," she said.' # Result in Python shell
>>> print('"Isn\'t," she said.') # Result from print()
"Isn't," she said.

>>> s = 'First line.\nSecond line.' # \n means newline
>>> s # without print(), \n is included in the output
'First line.\nSecond line.'

>>> print(s) # with print(), \n produces a new line
First line.
Second line.

```

การพิมพ์ข้อความกับ Python shell เข้าไปตรงๆ นั่น บางคำสั่งอาจจะแสดงผลลัพธ์ที่ไม่ถูกต้อง จากตัวอย่างด้านบน 'First line.\nSecond line.' สังเกตว่า ต้องการสั่งให้ขึ้นบรรทัดใหม่ด้วย \n แต่การทำงานนั้นไม่ถูกต้องเมื่อสั่งงานผ่าน Python shell ถ้าต้องการให้คำสั่งแสดงผลอย่างถูกต้องจะต้องใช้ควบคู่กับคำสั่ง print

Escape characters และ String formatting operator เป็นรหัสคำสั่งที่ใช้ควบคุมการแสดงผลร่วมกับคำสั่ง print ผู้เขียนได้อธิบายไว้แล้วในบทที่ 3 แต่ในไฟร์ซอฟต์แวร์ชั้น 3 ขึ้นไป จะมีรหัสควบคุมพิเศษเพิ่มเติม ดังในตารางที่ 4.1

ตารางที่ 4.1 Escape Characters

Backslash notation	Description
\cx	คำสั่งพิมพ์ Control และ x พร้อมกัน
\C-x	คำสั่งพิมพ์ Control และ x พร้อมกัน

\M-\C-x	คำสั่งพิมพ์ Meta, Control และ x พร้อมกัน
\t\t\t	พิมพ์เลขฐานแปด โดยที่ n มีค่าระหว่าง 0 - 7
\s	คำสั่งพิมพ์ช่องว่าง
\x	คำสั่งพิมพ์ตัวอักษร x
\xnn	พิมพ์เลขฐานสิบหก โดยที่ n มีค่าระหว่าง 0 – 9 และ A - F

ตัวดำเนินการพิเศษเกี่ยวกับสตริง (String special operators)

ตารางที่ 4.2 เป็นตัวดำเนินการพิเศษที่ใช้ทำงานร่วมกับสตริง เพื่อสามารถทำงานของตัวดำเนินการพิเศษดังกล่าว สมมุติให้ตัวแปร a มีค่าเท่ากับ 'Hello' และตัวแปร b เท่ากับ 'Python'

ตารางที่ 4.2 String special operators

Operator	Description	Example
+	เชื่อมสตริง 2 ชุดเข้าด้วยกัน	a + b = HelloPython
*	ทำสำเนาซ้ำจำนวนเท่ากับ n ตัว (*n)	a*3 = HelloHelloHello
[]	เข้าถึงข้อมูลสตริง ด้วยการระบุตำแหน่ง	a[1] = H, b[0] = P
[:]	เข้าถึงช่วงข้อมูลสตริง ด้วยการระบุตำแหน่งเริ่มต้น : ตำแหน่งสิ้นสุด	a[1:4] = ell
in	เป็นจริง เมื่อข้อมูลที่ทดสอบเป็นสมาชิก ในสตริง	'H' in a = True, 'L' in a = False
not in	เป็นจริง เมื่อข้อมูลที่ทดสอบไม่เป็นสมาชิกในสตริง	'H' not in a = False, 'L' not in a = True
%	ใช้จัดรูปแบบของสตริง	อ่านเพิ่มเติมในบทที่ 3: String formatting
r/R	เป็นการจัดรูปแบบสตริง ในรูปแบบที่ผู้ใช้กำหนดเอง โดยไม่ต้องสนใจ Escape character	print(r'\n') = \n print ('C:\\nowhere') = C:\\nowhere (ผิด) แต่ถ้าพิมพ์กับ r/R จะถูกต้อง print (r'C:\\nowhere') = C:\\nowhere (ถูก)

โดยปกติ สตริงในไฟรอนจะเป็นชนิด ASCII ที่มีขนาด 8 บิต ซึ่งไม่ครอบคลุมตัวอักษรของภาษาต่างๆ ทั่วโลก ดังนั้นมีผู้เขียนโปรแกรมต้องการใช้อักษรพิเศษอื่นๆ ที่เกินขอบเขตของ ASCII จะต้องใช้ตัวอักษรแบบ Unicode ซึ่งมีขนาด 16 บิต โดยใช้ตัวอักษร u นำหน้าข้อความ ดังนี้

```
>>> print (u'ทดสอบ Unicode')
```

ทดสอบ Unicode



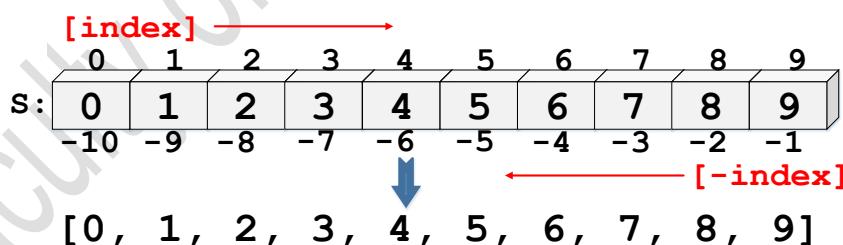
Note: ยูนิโคด (Unicode) คือ รหัสคอมพิวเตอร์ที่ใช้แทนตัวอักษร ตัวเลข, สัญลักษณ์ต่างๆ ได้มากกว่ารหัสแบบเก่าอย่าง ASCII ซึ่งเก็บตัวอักษรได้สูงสุดเพียง 256 ตัวอักษร โดย Unicode รุ่นปัจจุบันสามารถเก็บตัวอักษรได้ถึง 34,1683 ตัว จากรากภาษา ทั่วโลก 24 ภาษา โดยไม่สนใจว่าเป็นแพลตฟอร์มใด และไม่ขึ้นกับโปรแกรมใดๆ

2. ข้อมูลเชิงประกอบ (Composite data types)

คือข้อมูลที่ผู้เขียนโปรแกรมสามารถสร้างขึ้นมาโดยการผสม หรือประกอบจากชนิดข้อมูลที่แตกต่างกันๆ ได้ เช่นมาใช้งานได้เอง สามารถเก็บข้อมูลได้โดยที่ไม่จำเป็นต้องเป็นชนิดเดียวกัน เช่น `data = {'name': 'John', 'age': 25, 'city': 'New York'}` เป็นต้น ข้อมูลเชิงประกอบมีหลายชนิดประกอบไปด้วย ลิสต์ (Lists) ทัพเพิล (Tuples) ดิกชันนารี (Dictionaries) และเซต (Set)

ลิสต์ (lists)

ตัวแปรชนิดลิสต์ (list) คือ ตัวแปรที่สามารถเก็บข้อมูลได้หลายจำนวนต่อเนื่องกันภายใต้ตัวแปรเดียวกัน (สามารถเก็บข้อมูลต่างชนิดกันได้) มีลักษณะคล้ายกับอารเรย์ (Array ใช้เก็บข้อมูลชนิดเดียวกันเท่านั้น) ในโปรแกรมรุ่นเก่า อย่างเช่น C/C++ แต่มีการใช้งานที่ยืดหยุ่นกว่า เป็นตัวแปรที่เกิดขึ้นในภาษาใหม่ๆ การเข้าถึงข้อมูลภายใต้ลิสต์จะต้องระบุด้วยดัชนีลำดับของข้อมูล (ตำแหน่งที่เก็บข้อมูล) ที่เก็บเอาไว้ โดยเริ่มต้นจาก 0 เช่นเดียวกับอารเรย์ แต่ลิสต์มีความสามารถใช้ดัชนีที่เป็นค่าลบ (เข้าถึงข้อมูลจากด้านหลังของลิสต์) ได้ นั่นคือ ถ้าเป็น -1 หมายถึง ข้อมูลลำดับท้ายสุดของลิสต์ ตัวแปรชนิดลิสต์ใช้สัญลักษณ์ [] ในการเก็บข้อมูล และแยกข้อมูลแต่ละตัวด้วยสัญลักษณ์ , ดังตัวอย่างในรูปที่ 4.1



รูปที่ 4.1 โครงสร้างตัวแปรแบบลิสต์ (list)

จากรูปที่ 4.1 แสดงโครงสร้างของลิสต์ จากรูปสังเกตว่าตำแหน่งสำหรับเข้าถึงข้อมูลด้านซ้าย (index) จะเริ่มจาก 0 ไปจนถึง n แต่เมื่อต้องการเข้าถึงข้อมูลจากส่วนท้ายของลิสต์ทำได้โดยการอ้างตำแหน่งโดยใช้ค่าลบ (-index) ซึ่งข้อมูลในลำดับที่อยู่ท้ายสุดจะเป็น -1 เมื่อต้องการเข้าถึงข้อมูล 4 สามารถเข้าถึงได้ 2 แบบคือ [4] และ [-6] สำหรับการเก็บข้อมูลจริงเมื่อทำการเขียนโปรแกรม ตัวแปรลิสต์จะอยู่ภายใต้เครื่องหมาย [] และแยกข้อมูลด้วย , ดังรูปที่ 4.1 การประกาศตัวแปรลิสต์ใช้คำสั่งดังนี้

```
>>> lst = [1, 2, 3, 4, 'Foo', 5, 'Bar']
>>> print(lst)
[1, 2, 3, 4, 'Foo', 5, 'Bar']
>>> type(lst)
<class 'list'>
```

การประกาศตัวแปรสามารถกำหนดชนิดข้อมูลที่แตกต่างกันได้ เช่น

```
list0 = []           # Empty list
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
list4 = ['name', "Surname", 123, 0.5,[list1]]
a = (1,2,3)          #tuple data type
b = {1:'a',2:'b'}    #dictionary data type
c = [1, a, b]
```

การเข้าถึงข้อมูลสามารถเข้าถึงได้โดยใช้ [index], [-index] หรือเข้าถึงเป็นช่วงข้อมูลโดยใช้ [index_start: index_stop] หรือ [index_start:[index_stop[:step]] ดังตัวอย่างต่อไปนี้

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7];
print ("list1[0]:", list1[0])
print ("list1[-1]:", list1[-1])
print ("list2[3]:", list2[3])
print ("list2[-4]:", list2[-4])
print ("list2[1:5]:", list2[1:5])
print ("list2[::2]:", list2[::2])
print ("list2[2::2]:", list2[2::2])
print ("list2[2:7:2]:", list2[2:7:2])
print ("list2[:7]:", list2[:7])
print ("list2[4:]:", list2[4:])
```



OUTPUT

```
list1[0]: physics
list1[-1]: 2000
list2[3]: 4
list2[-4]: 4
list2[1:5]: [2, 3, 4, 5]
list2[::2]: [1, 3, 5, 7]
list2[2::2]: [3, 5, 7]
list2[2:7:2]: [3, 5, 7]
list2[:7]: [1, 2, 3, 4, 5, 6, 7]
list2[4:]: [5, 6, 7]
```

จากตัวอย่างผู้เขียนโปรแกรมสามารถเข้าถึงข้อมูลของลิสต์ได้หลายลักษณะ เมื่ออนสตริง (สามารถกลับไปอ่านวิธีการเข้าถึงข้อมูลอย่างละเอียดในหัวข้อ ตัวแปรแบบสายอักขระ หรือสตริง เพิ่มเติม) เช่น เข้าถึงข้อมูลจากส่วนหน้า จากส่วนหลัง เข้าถึงแบบเจาะจง หรือเข้าถึงเป็นช่วง เป็นต้น จากตัวอย่างข้างต้น [0] หมายถึงเข้าถึงข้อมูลในตำแหน่งแรกตรงส่วนหน้าของลิสต์, [-1] คือเข้าถึงจากส่วนหลังตำแหน่งแรก, [1:5] คือเข้าถึงข้อมูลตั้งแต่ตำแหน่งที่ 1 ถึง 5 จากส่วนหน้าลิสต์, [::2] คือเข้าถึง

ข้อมูลจากส่วนหัวตั้งแต่ตำแหน่งแรกถึงท้ายสุดของลิสต์ โดยกระโดดเป็นช่วง ช่วงละ 2 ตำแหน่ง, [2::2] คือเข้าถึงข้อมูลตั้งแต่ตำแหน่งที่ 2 จากส่วนหัวของลิสต์ไปถึงตำแหน่งสุดท้ายของลิสต์ โดยกระโดดเป็นช่วง ช่วงละ 2 ตำแหน่ง, [2:7:2] คือเข้าถึงข้อมูลจากหัวลิสต์ตั้งแต่ตำแหน่งที่ 2 ไปจนถึงตำแหน่งที่ 7 โดยกระโดดเป็นช่วง ช่วงละ 2 ตำแหน่ง, [:7] คือการเข้าถึงข้อมูลจากหัวลิสต์ถึงตำแหน่งที่ 7 และ [4:] คือเข้าถึงข้อมูลเริ่มจากตำแหน่งที่ 4 ของหัวลิสต์ไปจนถึงค่าตัวสุดท้ายของลิสต์

การปรับปรุงแก้ไขข้อมูลในลิสต์ สามารถทำได้โดยอ้างถึงตำแหน่งข้อมูลที่ต้องการแก้ไข โดยใช้ [index] หรือ [-index] เช่น

```
list = ['physics', 'chemistry', 1997, 2000];
print ("Old value available at index 2 : ",list[2])
list[2] = 2001;
print ("New value available at index 2 : ",list[2])
```


OUTPUT

Old value available at index 2 : 1997
 New value available at index 2 : 2001

การลบข้อมูลออกจากลิสต์ทำได้โดยใช้คำสั่ง del และตามด้วยตำแหน่งスマชิกที่ต้องการจะลบ [index] หรือ [-index] เช่น

```
list1 = ['physics', 'chemistry', 1997, 2000];
print ("Before deleting index 2: ",list1);
del list1[2];    # deleting command
print ("After deleting index 2: ",list1);
```


OUTPUT

Before deleting index 2: ['physics', 'chemistry', 1997, 2000]
 After deleting index 2: ['physics', 'chemistry', 2000]

ตัวดำเนินการพื้นฐานของลิสต์ (Basic list operations)

สำหรับตัวแปรแบบลิสต์สามารถใช้สัญลักษณ์ * และ + เช่นเดียวกับสตริงได้ ในลักษณะการเชื่อมค่าของスマชิกเข้าด้วยกัน (+) และการทำซ้ำข้อมูล (*) สำหรับตัวอย่างของการใช้งานตัวดำเนินการพื้นฐานมีดังต่อไปนี้

สมมุติให้ lst1 = [1, 2, 3] และ lst2 = [4, 5, 6]

```
len(lst1) = 3          # จำนวนของスマชิก หรือความยาวของ lst1 เท่ากับ 3
lst1 + lst2 = [1, 2, 3, 4, 5, 6]  # list concatenation
lst1 * 3 = [1, 2, 3, 1, 2, 3, 1, 2, 3]
3 in lst1      # 3 อยู่ใน lst1 หรือไม่ คำตอบคือ จริง (True)
for x in lst1: print (x)  # พิมพ์ข้อมูลスマชิกใน lst1 โดยใช้ for คำตอบคือ 1, 2, 3
```

ตัวอย่างการเขียนโปรแกรมดังนี้

```
lst1 = [1, 2, 3]; lst2 = [4, 5, 6]
print("length of lst1 :", len(lst1))
print("lst1 + lst2 :", lst1 + lst2)
print("lst1 * 3 :", lst1 * 3)
print("Elements in lst1 are :")
for x in lst1:
    print (x)
```



OUTPUT

```
length of lst1 : 3
lst1 + lst2 : [1, 2, 3, 4, 5, 6]
lst1 * 3 : [1, 2, 3, 1, 2, 3, 1, 2, 3]
Elements in lst1 are :
1
2
3
```

การเปรียบเทียบข้อมูลในตัวแปรลิสต์สามารถทำได้เหมือนสตริง โดยใช้ตัวดำเนินการคือ <, <=, >, >=, ==, !=, in และ not in ดังต่อไปนี้

```
lst1 = [1,2,3,4,5]
lst2 = [9,8,7,6,5]
lst3 = [9,8,7,6,5]
lst4 = [8,7]
print("lst1 < lst2 :", lst1 < lst2)
print("lst1 > lst2 :", lst1 > lst2)
print("lst2 >= lst1 :", lst2 >= lst1)
print("lst2 == lst3 :", lst2 == lst3)
print("8 in lst3 :", 8 in lst3)
print("lst4 not in lst2 :", lst4 not in lst2)
```



OUTPUT

```
lst1 < lst2 : True
lst1 > lst2 : False
lst2 >= lst1 : True
lst2 == lst3 : True
8 in lst3 : True
lst4 not in lst2 : True
```

การเปรียบเทียบในลิสต์จะเป็นการเปรียบเทียบในลักษณะค่าต่ำกว่า หรือสมาชิกต่อสมาชิก ที่จะตัวไปเรื่อยๆ ในกรณีแรก lst1 < lst2 โปรแกรมจะเริ่มตรวจสอบคุณของ lst1 และ lst2 คู่แรกก่อน ในที่นี่คือ 1 (lst1[0]) กับ 9 (lst2[0]) ผลที่ได้คือ 1 < 9 จริง ดังนั้นผลลัพธ์ที่ได้จะเป็นจริง (True) สำหรับตัวดำเนินการแบบ in และ not in จะเป็นการตรวจสอบสมาชิกที่อยู่ภายใต้ลิสต์ว่ามีอยู่หรือไม่ จากตัวอย่าง 8 อยู่ใน lst3 ดังนั้นผลลัพธ์ที่ได้คือ เป็นจริง

การแสดงผลข้อมูลในลิสต์ที่มีความยาวมากๆ นิยมใช้การทำซ้ำช่วยในการแสดงผล (ซึ่งจะกล่าวในบทที่ 6) ในส่วนนี้จะขอแนะนำการใช้ for สำหรับแสดงผลข้อมูลดังนี้

```
Sum = 0;
for i in [2,3,5,7,11,13,17,19]:
```

```
Sum += i
print ("Summation of list :",Sum)
```



Summation of list : 77

OUTPUT

จากตัวอย่างเป็นการหาผลรวมของสมาชิกทั้งหมดในลิสต์ โดยใช้ for ในการวนลูบซ้ำ เพื่อดึงค่าข้อมูลในลิสต์มาทีละตัว และทำการหาผลรวมของสมาชิกทั้งหมดเอาไว้ในตัวแปร Sum คำสั่ง for จะวนซ้ำทั้งหมด 8 ครั้งจึงหยุดทำงาน ผลลัพธ์ที่เกิดจากผลรวมของสมาชิกในลิสต์มีค่าเท่ากับ 77

เมื่อต้องการลบข้อมูลตัวใดตัวหนึ่งของสมาชิกภายในลิสต์ หรือลบตัวแปรลิสต์ทั้งลิสต์ สามารถทำได้โดยใช้คำสั่ง del และตามด้วยตำแหน่งสมาชิก หรือตัวแปร เช่น

```
lst = [i*1 for i in range(10)]
print("list before removing member:",lst)
del lst[0], lst[2], lst[4], lst[6]
print("list after removing member:",lst)
del lst
print("list after removing member:",lst)
```



```
list before removing member: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list after removing member: [1, 2, 4, 5, 7, 8]
Traceback (most recent call last):
  File "C:\Python34\testx.py", line 6, in <module>
    print("list after removing member:",lst)
NameError: name 'lst' is not defined
```

จากโปรแกรมตัวอย่างด้านบน คำสั่ง [*i**1 for *i* in range(10)] คือการสร้างลิสต์โดยใช้ for ทำงานร่วมกับนิพจน์คณิตศาสตร์ โดยค่า *i* จะมีค่าตั้งแต่ 0 – 9 จากนั้นนำค่า *i* แต่ละค่ามาคูณกับ 1 ทำให้ได้ผลลัพธ์เป็นตัวแปรลิสต์ที่มีสมาชิกเป็น [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] ต่อจากนั้นทดสอบลบข้อมูลโดยใช้คำสั่ง del lst[0], lst[2], lst[4], lst[6] ส่งผลให้สมาชิกที่ตำแหน่ง 0, 2, 4 และ 6 ถูกลบออกไป เมื่อทำการลบตัวแปรด้วยคำสั่ง del lst และทดสอบพิมพ์ค่าข้อมูลที่อยู่ภายใต้ตัวแปรดังกล่าวอีกครั้ง ผลที่ได้คือ ไฟรอนจะแสดงข้อผิดพลาดออกมาเนื่องจากตัวแปรดังกล่าวได้ถูกลบออกไปจากหน่วยความจำเรียบร้อยแล้วนั่นเอง

ทัพเพิล (Tuples)

ทัพเพิลมีลักษณะโครงสร้างคล้ายกับลิสต์ คือ สามารถเก็บข้อมูลได้ในปริมาณมาก และสามารถเก็บข้อมูลต่างประเภทกันได้ภายในตัวแปรเดียว กัน ข้อแตกต่างระหว่างลิสต์กับทัพเพิลคือ ทัพเพิลจะใช้สำหรับเก็บข้อมูลที่มีค่าคงที่ และไม่มีการเปลี่ยนแปลง คล้ายอะเรย์ที่มีขนาดคงที่ ไม่สามารถเพิ่มหรือลบข้อมูลทัพเพิลได้โดยตรง แต่ทำให้มีข้อได้เปรียบในเรื่องของความเร็วในการเข้าถึงข้อมูล สำหรับการ

เข้าถึงข้อมูลทำได้โดยใช้ตัวชี้ หรือดัชนีเหมือนลิสต์ ตัวแปรทัพเพิลจะใช้สัญลักษณ์ (...) ในการประกาศ
ตัวแปร และสามารถใช้ในทัพเพิลจะคืนด้วย , ดังตัวอย่าง

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
tup3 = (19, 12.5, 'Python', "HELLO");
tup4 = (tup1, tup2, tup3);
tup5 = tup1 + tup2
print(tup1)
print(tup2)
print(tup3)
print(tup4)
print(tup5)
```



OUTPUT

```
(12, 34.56)
('abc', 'xyz')
(19, 12.5, 'Python', 'HELLO')
((12, 34.56), ('abc', 'xyz'), (19, 12.5, 'Python', 'HELLO'))
(12, 34.56, 'abc', 'xyz')
```

ให้คำสั่ง type (ตัวแปร) เพื่อตรวจสอบชนิดของข้อมูล ด้วยอย่างเช่น

```
>>> tup1 = (12, 3.5, 'Hi');
>>> type(tup1) # Display class type
<class 'tuple'>
>>> tup1 # Display structure of Tuple
(12, 3.5, 'Hi')
```

เมื่อผู้เขียนโปรแกรมต้องการกำหนดค่าเริ่มต้นให้กับทัพเพิล เป็นค่าว่าง มีรูปแบบดังนี้

```
tup1 = ();
print ("Tuple empty : ",tup1)
```



OUTPUT

```
Tuple empty : ()
```

การกำหนดค่าสมาชิกให้กับทัพเพิลเพียงค่าเดียว สามารถทำได้ 2 รูปแบบ ดังนี้

```
tup1 = (30);
tup2 = (30,);
print("Tuple 1 :",tup1);
print("Tuple 2 :",tup2);
```



OUTPUT

```
Tuple 1 : 30
Tuple 2 : (30,)
```

เมื่อกำหนดค่าให้กับตัวแปรทัพเพิล มากกว่า 2 ตัวขึ้นไปพร้อมกัน จำนวนของข้อมูลทางด้าน

ขวา มีอ ต้องมีค่าเท่ากับตัวแปรทางด้านซ้ายมีอ เช่น

```
>>> tup1, tup2 = (1, 2);
>>> tup1
1
```

```
>>> tup2
2
```

จากโปรแกรมข้างต้นจะพบว่า tup1 เก็บค่า 1 และ tup2 จะเก็บค่า 2 ไว้ ซึ่งจะแตกต่างจากการกำหนดค่าของตัวแปรทั่วๆ ไป เพราะการกำหนดตัวแปรแบบทั่วไป tup1 และ tup2 ควรจะมีค่าเท่ากับ (1, 2) เมื่อongกัน ลองทดสอบการกำหนดค่าใหม่อีกรอบด้วยรูปแบบต่อไปนี้

```
tup1, tup2 = (1, 2), (3, 4);
>>> tup1
(1, 2)
>>> tup2
(3, 4)

tup1, tup2 = ((1, (2, (3, 4))), (((5, 6), 7), 8))
>>> tup1
(1, (2, (3, 4)))
>>> tup2
(((5, 6), 7), 8)
>>>
```

ไฟชอนใช้ , ในการแยกค่าข้อมูลที่อยู่ทางด้านขวาเมื่อ เพื่อกำหนดให้ตัวแปรที่อยู่ทางซ้ายเมื่อ ได้อย่างถูกต้อง และควรจำไว้ว่า ต้องกำหนดจำนวนของค่าข้อมูลให้เท่ากับตัวแปรเสมอ มิเช่นนั้นจะทำให้เกิดข้อผิดพลาดขึ้น เช่น

```
>>> tup1, tup2 = (1, 2, 3)
tup1, tup2 = (1, 2, 3)
ValueError: too many values to unpack (expected 2)
```

ในบางสถานะการณ์ผู้เขียนโปรแกรมมีความต้องการกำหนดตัวแปรทัพเพิล ให้สามารถเก็บค่าข้อมูลคล้ายดิกชันnaire มีรูปแบบดังนี้

```
>>> tup1 = (("Name", "Michael"), ("Age", "35"), ("Sex", "Male"))
>>> tup1[0]
('Name', 'Michael')
>>> tup1[1]
('Age', '35')
```

การแก้ไขข้อมูลหลังจากประกาศตัวแปรแล้วเป็นข้อห้ามของทัพเพิล ลองทดสอบเปลี่ยนค่าข้อมูลในทัพเพิล โดยการอัปเดตข้อมูลใน tup1[0] ดังตัวอย่าง

```
tup1 = (12, 34.56);
tup1[0] = 15
```


OUTPUT

```
Traceback (most recent call last):
  File "C:/Python33/testx.py", line 2, in <module>
    tup1[0] = 15
TypeError: 'tuple' object does not support item assignment
```

จากตัวอย่าง โปรแกรมข้างต้น จะเกิดข้อผิดพลาดขึ้น เนื่องจาก tuples จะไม่อนุญาตให้เปลี่ยนแปลง หรือแก้ไขข้อมูลใดๆ หลังจากที่มีประกาศตัวแปรแล้ว

สำหรับการลบตัวแปรแบบทัพเพิลสามารถกระทำได้โดยใช้คำสั่ง `del` และตามด้วยชื่อตัวแปร

```
tup = ('physics', 'chemistry', 1997, 2000);
print (tup);
del tup;
print ("After deleting tup : ",tup)
```



OUTPUT

```
('physics', 'chemistry', 1997, 2000)
Traceback (most recent call last):
  File "C:/Python33/testx.py", line 4, in <module>
    print ("After deleting tup : ",tup)
NameError: name 'tup' is not defined
```

เมื่อสั่งรันโปรแกรม ในบรรทัดที่ 2 สามารถพิมพ์ข้อมูลของ `tup` ได้ แต่เมื่อใช้คำสั่งลบตัวแปรด้วย `del` และทำการสั่งพิมพ์ตัวแปรดังกล่าวอีกครั้งจะเกิดข้อผิดพลาดขึ้น เพราะตัวแปรดังกล่าวถูกลบออกจากหน่วยความจำของเครื่องไปแล้วนั่นเอง

ตัวดำเนินการพื้นฐานที่ใช้กับทัพเพิล (Basic tuples operations)

สำหรับตัวแปรแบบทัพเพิลจะสามารถใช้สัญลักษณ์ * และ + เช่นเดียวกับลิสต์ โดย (+) ใช้สำหรับเชื่อมสมาชิกระหว่างทัพเพิลเข้าด้วยกัน และใช้ (*) สำหรับทำซ้ำข้อมูลสมาชิก สำหรับตัวอย่างของการใช้งานตัวดำเนินการพื้นฐานมีดังต่อไปนี้

สมมุติให้ `tup1 = (1, 2, 3)` และ `tup2 = (4, 5, 6)`

```
len(tup1) = 3          # จำนวนของสมาชิก หรือความยาวของ tup1 เท่ากับ 3
```

```
tup1 + tup2 = (1, 2, 3, 4, 5, 6) # tuple concatenation
```

```
tup1 * 3 = (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

```
3 in tup1           # 3 อยู่ใน tup1 หรือไม่ คำตอบคือ จริง (True)
```

```
for x in tup1: print (x) # พิมพ์ข้อมูลสมาชิกใน tup1 โดยใช้ for คำตอบคือ 1, 2, 3
```

ตัวอย่างการโปรแกรมใช้งานตัวแปรทัพเพิล ดังนี้

```
tup1 = (1, 2, 3); tup2 = (4, 5, 6)
print("length of tup1 :",len(tup1))
print("tup1 + tup2 :",tup1 + tup2)
print("tup1 * 3 :",tup1*3)
print("Elements in tup1 are :")
for x in tup1:
    print (x)
```


OUTPUT

```
length of tup1 : 3
tup1 + tup2 : (1, 2, 3, 4, 5, 6)
tup1 * 3 : (1, 2, 3, 1, 2, 3, 1, 2, 3)
Elements in tup1 are :
1
2
3
```

การเข้าถึงข้อมูลในตัวแปรชนิดทัพเพิล เหมือนกับตัวแปรชนิดลิสต์ ดังตัวอย่างต่อไปนี้

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7);
print ("tup1[0]:", tup1[0])
print ("tup1[-1]:", tup1[-1])
print ("tup2[3]:", tup2[3])
print ("tup2[-4]:", tup2[-4])
print ("tup2[1:5]:", tup2[1:5])
print ("tup2[::-2]:", tup2[::-2])
```


OUTPUT

```
tup1[0]: physics
tup1[-1]: 2000
tup2[3]: 4
tup2[-4]: 4
tup2[1:5]: (2, 3, 4, 5)
tup2[::-2]: (1, 3, 5, 7)
```

การเปรียบเทียบข้อมูลในตัวแปรทัพเพิลสามารถทำได้เหมือนสตริง โดยใช้ตัวดำเนินการคือ <, <=, >, >=, ==, !=, in และ not in ดังต่อไปนี้

```
tup1 = (1,2,3,4,5)
tup2 = (9,8,7,6,5)
tup3 = (9,8,7,6,5)
tup4 = (8,7)
print("Tup1 < Tup2 :", tup1 < tup2)
print("Tup1 > Tup2 :", tup1 > tup2)
print("Tup2 >= Tup1 :", tup2 >= tup1)
print("Tup2 == Tup3 :", tup2 == tup3)
print("8 in Tup3 :", 8 in tup2)
print("Tup4 not in Tup2 :", tup4 not in tup2)
```


OUTPUT

```
Tup1 < Tup2 : True
Tup1 > Tup2 : False
Tup2 >= Tup1 : True
Tup2 == Tup3 : True
8 in Tup3 : True
Tup4 not in Tup2 : True
```

การเปรียบเทียบในทัพเพิลจะเป็นการเปรียบเทียบในลักษณะค่าต่อค่า หรือสมาชิกต่อสมาชิก ที่จะตัวไปเรื่อยๆ ในกรณีแรก Tup1 < Tup2 โปรแกรมจะเริ่มตรวจสอบคุณของ Tup1 และ Tup2 คู่แรกก่อน ในที่นี้คือ 1 (Tup1[0]) กับ 9 (Tup2[0]) ผลที่ได้คือ 1 < 9 จริง ดังนั้นผลลัพธ์ที่ได้จะเป็นจริง (True)

สำหรับตัวดำเนินการแบบ in และ not in จะเป็นการตรวจสอบสมาชิกที่อยู่ภายในทัพเพิลว่ามีอยู่หรือไม่ จากตัวอย่าง 8 อยู่ใน Tup3 ดังนั้นผลลัพธ์ที่ได้คือ เป็นจริง

ดิกชันนารี (Dictionary)

ดิกชันนารีเป็นตัวแปรชนิดเปลี่ยนแปลงข้อมูลได้ตลอดเวลา (Mutable) คล้ายกับลิสต์ และมีลักษณะการเก็บข้อมูลที่เหมือนกับลิสต์และทัพเพิล คือสามารถเก็บข้อมูลได้หลายค่า และต่างชนิดกันได้พร้อมๆ กัน คุณสมบัติพิเศษที่แตกต่างของดิกชันนารีคือ จะเก็บข้อมูลเป็นคู่ คือคีย์ (Key) ซึ่งจะต้องไม่ซ้ำกัน กับข้อมูล (Value) ซึ่งข้อมูลทั้งสองต้องมีความสัมพันธ์กัน โดยที่คีย์เปรียบเหมือนตัวชี้ (Identification) เพื่ออ้างอิงไปยังข้อมูลจริงที่ต้องการใช้งาน การประกาศตัวแปรดิกชันนารีจะใช้สัญลักษณ์ { ... } ในการประกาศตัวแปร สำหรับการเก็บข้อมูลคู่ของสมาชิกจะใช้สัญลักษณ์ : เช่น ระหว่างคีย์และข้อมูล และใช้สัญลักษณ์ , ในการแยกแยะระหว่างสมาชิกในดิกชันนารี สำหรับการเข้าถึงข้อมูลของดิกชันนารีจะใช้สัญลักษณ์ [...] เมื่อกับลิสต์ และทัพเพิล ดังตัวอย่างต่อไปนี้

```
dict1 = {}          #empty dictionary
dict2 = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
dict3 = { 'abc': 456 };
dict4 = { 'abc': 123, 98.6: 37 };
print(dict1)
print(dict2)
print(dict3)
print(dict4)
```



OUTPUT

```
{ }
{'Alice': '2341', 'Cecil': '3258', 'Beth': '9102'}
{'abc': 456}
{98.6: 37, 'abc': 123}
```



Note: ควรจำไว้ว่า คีย์ (Key) จะต้องมีค่าที่ไม่ซ้ำกัน (Unique) และเป็นชนิดของไร์ก์ได้ แต่มีข้อแม้ว่า ต้องเป็นข้อมูลประเภทที่ไม่เปลี่ยนแปลง (Immutable) เช่น สตริง เลขจำนวน หรือทัพเพิล

จากการประกาศตัวแปรดิกชันนารีข้างต้น ตัวแปร dict1 ประกาศให้มีสมาชิกว่าง เมื่อแสดงผลจะปรากฏสัญลักษณ์ { } แสดงถึงไม่มีสมาชิกใดๆ เก็บอยู่ในตัวแปรดังกล่าว ในตัวแปร dict2 ได้ประกาศชนิดของตัวแปรที่เก็บหลายประเภทในตัวแปรดิกชันนารีตัวเดียว กัน โดยสมาชิกตัวแรกจะมีคีย์คือ Alice เพื่อซื้อป้ายข้อมูลจริงคือ 2341 คันด้วยเครื่องหมาย : และ Cecil เป็นคีย์ซื้อป้ายข้อมูล 3258 เป็นต้น สำหรับ dict3, dict4 นั้นเป็นการประกาศตัวแปรโดยใช้ ; ปิดท้ายคำสั่ง (เมื่อในภาษาซี)

การเข้าถึงข้อมูลในดิกชันนารีสามารถเข้าถึงได้โดยใช้เครื่องหมาย [...] ร่วมกับชื่อคีย์ เมื่อกับตัวแปรลิสต์ และทัพเพิล เช่น



```
dict = { 'Name': 'Zara', 'Age': 7, 'Class': 'First' };
print ("dict['Name']: ", dict['Name']);
print ("dict['Age']: ", dict['Age']);
```



OUTPUT

```
dict['Name']: Zara
dict['Age']: 7
```

จากตัวอย่างด้านบน เมื่อการต้องการเข้าถึงข้อมูลชื่อ Zara ให้อ้างอิงไปที่ตัวแปรชื่อ dict โดยใช้คีย์คือ Name ร่วมกับเครื่องหมาย [..] ดังนี้ dict['name'] เมื่อผู้เขียนโปรแกรมพยายามที่จะอ้างอิงไปยังข้อมูลที่ไม่มีคีย์ปรากฏอยู่ในตัวแปรดิกชันนารี จะทำให้เกิดข้อผิดพลาดขึ้น



Tips: คีย์ (key) สามารถใช้ค่าว่างได้ เช่น dict1 = {'':111, ''':222} แต่ไม่แนะนำให้กระทำ เพราะคีย์ควรสื่อความหมาย เพื่อใช้อ้างอิงถึงข้อมูลที่มีความสัมพันธ์กัน

ในกรณีที่ผู้เขียนโปรแกรมยังไม่ทราบข้อมูลที่จะใส่ลงไปในดิกชันนารีแนชัดในเบื้องต้น แต่วางแผนไว้ว่าจะใส่ข้อมูลในภายหลัง ให้กำหนดเป็นดิกชันนารีว่างเตรียมไว้ก่อน และจึงปรับปรุงข้อมูลภายหลัง เช่น

```
dict1 = {};
print("Original dict : ",dict1)
dict1['Key1'] = 123;
print("Update data1 to dict : ",dict1)
dict1['Key2'] = 456;
print("Update data2 to dict : ",dict1)
```



OUTPUT

```
Original dict : {}
Update data1 to dict : {'Key1': 123}
Update data2 to dict : {'Key1': 123, 'Key2': 456}
```

จากตัวอย่างโปรแกรมข้างบน เป็นการปรับปรุงข้อมูลในดิกชันนารีโดยการเพิ่มคู่ของคีย์และข้อมูลให้ดิกชันนารีเข้าไปได้เรียบร้อย มีข้อแม้ว่าต้องมีการประกาศตัวแปรดิกชันนารีแบบว่างเปล่าไว้ก่อนเสมอ คือ dict1 = {} ต่อจากนั้นจึงสามารถเพิ่มข้อมูลได้ ในการใช้งานดิกชันนารีขั้นสูงนั้น ผู้เขียนโปรแกรมสามารถสร้างคีย์ และข้อมูล ได้หลายรูปแบบ เช่น คีย์หนึ่งคู่กับข้อมูลหนึ่งค่า คีย์หนึ่งคู่กับข้อมูลหลายค่า คีย์หลายคู่กับข้อมูลหนึ่งค่า และคีย์หลายคู่กับข้อมูลหลายค่า ดังตัวอย่าง

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9); # tuples
chars = ('a', 'b', 'c', 'd', 'e', 'f'); # tuples
DummyDict = {}; # Empty dictionary
DummyDict['001'] = 'John'; #One to One Dict
print("One to One Dictionary : ", DummyDict)
DummyDict['001'] = numbers; #One to Many Dict
print("One to Many Dictionary : ", DummyDict)
DummyDict = {};
DummyDict[numbers] = 'Python'; #Many to One Dict
print("Many to One Dictionary : ", DummyDict);
DummyDict[numbers] = chars; #Many to Many Dict
print("Many to Many Dictionary : ", DummyDict)
```


OUTPUT

```
One to One Dictionary:{'001': 'John'}
One to Many Dictionary:{'001': (0,1,2,3,4,5,6,7,8,9)}
Many to One Dictionary:{(0,1,2,3,4,5,6,7,8,9): 'Python'}
Many to Many Dictionary : {(0,1,2,3,4,5,6,7,8,9): ('a', 'b',
'c', 'd', 'e', 'f')}
```

การแก้ไขข้อมูลใน딕셔너รี่สามารถทำได้โดยการอ้างถึงด้วยคีย์ และกำหนดค่าโดยใช้

เครื่องหมาย '=' เช่น dict1['Key1'] = 'xyz' (กำหนดให้ Key1 ชี้ข้อมูล xyz) เป็นต้น แต่ถ้าไม่มีคีย์อยู่ในตัวแปร딕셔너รี่ที่ประกาศไว้ ไฟอนถือว่าเป็นการสร้างคุ่ของคีย์และข้อมูลใหม่ ตัวอย่างเช่น dict1 มีข้อมูลดังนี้

```
>>> dict1 = {'Key1': 123}
```

เมื่อทำการเพิ่มคุ่ของคีย์และข้อมูลเข้าไป (Key2: 345) โดยที่ dict1 ไม่มีข้อมูลชุดนี้อยู่ก่อนหน้าจะทำให้ dict1 มีข้อมูลดังนี้

```
>>> dict1['Key2'] = 456
>>> dict1
{'Key2': 456, 'Key1': 123}
```

การลบข้อมูลออกจาก딕셔너รี่ทำได้ใน 3 รูปแบบคือ ลบเฉพาะสมาชิกที่ต้องการ เคลียร์ค่าข้อมูลทั้งหมด หรือลบตัวแปร딕셔너รี่ออกจากหน่วยความจำ โดยใช้คำสั่ง del สำหรับการลบข้อมูลเฉพาะสมาชิกที่ต้องการทำได้โดยใช้รูปแบบคำสั่ง del ชื่อตัวแปร[คีย์] การเคลียร์ข้อมูลมีรูปแบบคือ ชื่อตัวแปร.clear() และการลบตัวแปร딕셔너รี่มีรูปแบบ del ชื่อตัวแปร ดังต่อไปนี้

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
print("Original dict :",dict);
del dict['Name']; # remove entry with key 'Name'
print("After removed 'Name' :",dict);
dict.clear();      # remove all entries in dict
print("After cleared all :",dict);
del dict;          # delete entire dictionary
print("After deleted from memory :",dict);
```


OUTPUT

```
Original dict : {'Class': 'First', 'Name': 'Zara', 'Age': 7}
After removed 'Name' : {'Class': 'First', 'Age': 7}
After cleared all : {}
After deleted from memory : <class 'dict'>
```

สามารถใน딕셔너รี่ สามารถเป็นได้ทั้งลิสต์และทัพเพิล กรณีที่สร้าง딕셔너รี่โดยมีคีย์เป็นทัพเพิลและข้อมูลสมาชิกเป็นสตริง ตัวอย่างเช่น

```
>>> Number = {(-1, 0, 1): 'Integer', (1.0, 0.0, -1.0): 'Float'}
>>> Number
{(-1, 0, 1): 'Integer', (1.0, 0.0, -1.0): 'Float'}
```

หรือกรณีที่สร้าง딕셔너รี่โดยที่มีคีย์เป็นสตริงและข้อมูลสมาชิกเป็นสตริง จำนวนเต็ม หรือลิสต์ เช่น

```
>>> myBoat = {"NAME":"KaDiMa", "LOA":18, "SAILS":["main","jib"]}
>>> myBoat
{'SAILS': ['main', 'jib'], 'NAME': 'KaDiMa', 'LOA': 18}
```

NAME, LOA และ SAILS คือคีย์ที่เป็นสตริง สำหรับข้อมูลสมาชิกที่เป็นสตริงคือ KaDiMa, จำนวนเต็มคือ 18 และลิสต์คือ ["main","jib"] ตามลำดับ

การแสดงผลข้อมูลใน딕ชันnaireที่มีความยาวมากๆ นิยมใช้คำสั่งทำซ้ำช่วยในการแสดงผล เช่น for เป็นต้น ดังตัวอย่างต่อไปนี้

```
dict1 = {1:"Python", 2:'Programming', 3:"Language", 4:"!!!!."}
for i in dict1:
    print(dict1[i])

dict2 = {1:"Python", 2:(1, 2), '3':[3, 4, 5], 4:{'Name':'Suchart'}}
for i in dict1:
    print(dict2[i])
```



OUTPUT

```
Python
Programming
Language
!!!!.
Python
(1, 2)
[3, 4, 5]
{'Name': 'Suchart'}
```

จากตัวอย่าง dict1 เป็นตัวแปรชนิดิกชันnaireที่มีคีย์เป็น จำนวนเต็ม และข้อมูลสมาชิกเป็นสตริง ในตัวอย่างเลือกใช้ for ในการแสดงผลข้อมูลสมาชิกทั้งหมด โดย การดึงคีย์ออกมาจาก dict1 ทีละค่า (for i in dict1) และใช้คีย์ดังกล่าว เข้าถึงข้อมูลสมาชิกที่เก็บจริง เช่น เมื่อคีย์เท่ากับ 1 คีย์ดังกล่าวจะถูกเก็บไว้ในตัวแปร i ต่อจากนั้นใช้ตัวแปร i เพื่ออ้างอิงไปยังข้อมูลจริงอีกรอบด้วย dict1[i] = dict1[1] = "Python" สำหรับการแสดงผลข้อมูลใน dict2 ก็จะทำในลักษณะเช่นเดียวกัน

คุณสมบัติของดิกชันnaireที่ควรต้องจดจำ

ประการแรก: ตามทฤษฎีแล้ว คีย์ในดิกชันnaireจะต้องมีค่าไม่ซ้ำกันเลย แต่ในสถานะการณ์จริง ไฟรอนยอมให้สามารถประกาศคีย์ที่เหมือนกันในดิกชันnaireได้ (ไฟรอนไม่ได้ตรวจสอบขณะประกาศข้อมูลในตัวแปร) ส่งผลให้การอ้างอิงข้อมูลที่มีคีย์เหมือนกันเกิดข้อผิดพลาดคือ ไฟรอนจะดึงข้อมูลรายการสุดท้ายที่เจอนในดิกชันnaireออกมาระบุและแสดงผล โดยข้อมูลที่ซ้ำกันในลำดับก่อนหน้าจะไม่ถูกนำมาใช้งานเลย ยกตัวอย่างเช่น

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'};
print ("dict['Name']: ", dict['Name']);
```



```
dict['Name']: Manni
```

OUTPUT

จากตัวอย่างโปรแกรมข้างบน แสดงให้เห็นว่ามีคีย์เหมือนกัน 2 ตำแหน่งในติกรชั้นนารี คือ Name เมื่อทำการเรียกใช้งานข้อมูล ปรากฏว่าโปรแกรมจะนำข้อมูลหลังสุดของติกรชั้นนารีมาแสดงผล เช่น ผลให้คุณของข้อมูล 'Name': 'Zara' จะไม่ถูกนำมาประมวลผลเลย

ประการที่สอง: โดยปกติคีย์จะต้องเป็นค่าที่ไม่ควรเปลี่ยนแปลง เพราะเป็นค่าที่ใช้อ้างอิงถึงข้อมูลจริง ตัวอย่างในชีวิตประจำวัน เช่น นักศึกษาต้องมีหมายเลข ID ของนักศึกษา และไม่ควรเปลี่ยน ID ไปเรื่อยๆ ฉันได้ก็ฉันนั้น คือไม่ควรเปลี่ยนแปลงบ่อยเช่นเดียวกัน แม้ว่าเราสามารถใช้ข้อมูลหลายประเภทเป็นคีย์ได้ เช่น สตริง ตัวเลข หรือทัพเพลิ แต่ไฟรอนไม่แนะนำให้ใช้คีย์ในลักษณะที่คีย์ถูกครอบด้วย [...] เพราะจะทำให้ไฟรอนเกิดความสับสนในการอ้างถึงข้อมูล และเกิดความผิดพลาด เช่น

```
dict = { ['Name']: 'Zara', 'Age': 7};
print ("dict['Name']: ", dict['Name']);
```



```
Traceback (most recent call last):
  File "C:/Python33/testx.py", line 1, in <module>
    dict = { ['Name']: 'Zara', 'Age': 7};
    ^
TypeError: unhashable type: 'list'
```

แต่ถ้าจำเป็นต้องการใช้ [...] เป็นคีย์จริงๆ สามารถทำได้โดยใช้เครื่องหมาย '' ในลักษณะดังนี้

'[KeyName]' แทน เช่น

```
dict = { '[Name]': 'Zara', 'Age': 7};
print ("dict['[Name]']: ", dict['[Name]]");
```



```
dict['[Name]']: Zara
```

เซต (Sets)

เซต ในทางคณิตศาสตร์เป็นค่าที่ใช้บ่งบอกถึงกลุ่มของสิ่งต่างๆ ว่า สิ่งใดอยู่ในกลุ่ม สิ่งใดไม่อยู่ในกลุ่ม เช่น เซตระนาภภาษาอังกฤษ หมายถึง กลุ่มของสารอังกฤษ a, e, i, o และ n เซตของจำนวนนับที่น้อยกว่า 10 หมายถึง กลุ่มตัวเลข 1, 2, 3, 4, 5, 6, 7, 8 และ 9 เป็นต้น สิ่งที่อยู่ในเซตเรียกว่า สมาชิก (Element หรือ Members) คุณสมบัติของเซตคือ สมาชิกไม่จำเป็นต้องเรียงลำดับ และสมาชิกต้องไม่ซ้ำกัน สำหรับการดำเนินงานกับเซตพื้นฐานประกอบไปด้วย การทดสอบความเป็นสมาชิก การจัดสมาชิก ที่ซ้ำกันทั้งหมดในเซต, union, intersection, difference และ symmetric difference เป็นต้น ในไฟรอนจะใช้สัญลักษณ์ {...} สำหรับสร้างตัวแปรชนิดเซต และใช้ set() ในการกำหนดว่าเป็นเซตว่าง ดังตัวอย่าง ต่อไปนี้

```
>>> setx = set();      # empty set
>>> type(setx);
<class 'set'>
```

จากตัวอย่างคำสั่งข้างบน เป็นการสร้างตัวแปร setx ให้เป็นเซตว่าง เมื่อเรียกคำสั่ง type(setx) จะแสดงประเภทของตัวแปร setx คือ set



Note: ควรจำไว้ว่า การประกาศตัวแปรเซตให้เป็นเซตว่าง จะต้องใช้ฟังชัน set() เท่านั้น ผู้ใช้ไม่สามารถใช้ setx = {} ได้ เพราะ Python จะมองว่าเป็นตัวแปรชนิดดิกชันนารี และไม่สามารถใช้รูปแบบ setx = () ได้ เพราะจะเป็นตัวแปรทัพเพิล และ setx = [] จะเป็นตัวแปรชนิดลิสต์แทน

ตัวอย่างการประกาศตัวแปรเซตที่ไม่ว่าง

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)      # show that duplicates have been removed
```



{'banana', 'orange', 'apple', 'pear'}

OUTPUT

จากโปรแกรมตัวอย่างข้างบน เป็นการประกาศตัวแปรชนิดเซต basket พร้อมกำหนดค่าของสมาชิก เป็น apple, orange, pear, banana เห็นได้ว่าสมาชิกที่กำหนดลงไปในเซตจะมีค่าที่ซ้ำกัน แต่เมื่อส่องรันโปรแกรมผลลัพธ์ที่ได้คือ เซตจะทำการกำจัดสมาชิกที่ซ้ำกันออก จึงเหลือแค่ 4 ตัวเท่านั้น

การสร้างเซตใน Python สามารถสร้างจากลิสต์ และทัพเพิลได้ โดยการประกาศตัวแปร และข้อมูลชนิดลิสต์ก่อน แล้วจึงเรียกฟังชันเพื่อเปลี่ยนค่าตัวแปรจากชนิดลิสต์มาเป็นเซต โดยใช้ฟังชัน set() ดังนี้

```
List_x = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
type(list_x)
<class 'list'> # Declear list_x was List

setList = set(list_x) # Converting List to Set
print(setList)
{'banana', 'orange', 'apple', 'pear'} # remove duplicate member
type(setList)
<class 'set'>

tuple_x = ("hello", "world", "of", "words", "of", "world")
type(setTuple)           # Declear tuple_x was Tuple
<class 'tuple'>
setTuple = set(tuple_x)   # Converting Tuple to Set
print(setTuple)
{'world', 'of', 'hello', 'words'}
```

ผู้เขียนโปรแกรมสามารถประกาศเซตจากตัวอักษร หรือตัวเลขที่ไม่ซ้ำกันได้ในครั้งเดียว โดยใช้รูปแบบคือ set('ตัวเลขหรือตัวอักษร') ดังนี้

```
setA = set('abracadabra')
setB = set('alacazam')
setC = set('112768730893488746249')
print(setA)
print(setB)
print(setC)
```



OUTPUT { 'b', 'r', 'a', 'd', 'c' }
{ 'z', 'c', 'a', 'l', 'm' }
{ '2', '3', '0', '1', '6', '7', '4', '8', '9' }

การตรวจสอบข้อมูลในเซต ทำได้โดยใช้คำสั่ง `in` ดังนี้

เมื่อ `setA` มีค่าเท่ากับ `{'b', 'r', 'a', 'd', 'c'}`

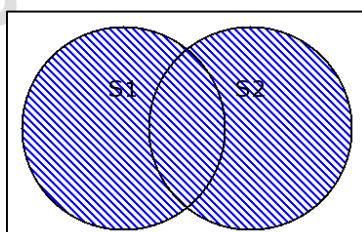
```
>>> 'a' in setA
True
>>> 'z' in setA
False
```

จากตัวอย่างโปรแกรมข้างต้น แสดงให้เห็นว่า '`a`' เป็นสมาชิกที่อยู่ใน `setA` และ '`z`' ไม่เป็นสมาชิกใน `setA`

ไฟชอนเตรียมคำสั่งไว้ให้ผู้ใช้งานสามารถดำเนินการเกี่ยวกับเซต เช่น `union (|)`, `intersection (&)`, `difference (-)`, `symmetric difference (^)` ไว้อย่างครบถ้วน ดังนี้

กำหนดให้ $S1 = \{1, 2, 3, 5, 8, 13\}$ และ $S2 = \{2, 3, 5, 7, 11, 13\}$

มุ่งเนี่ยน (**Union**) ของเซต $S1$ และ $S2$ คือเซตที่ประกอบด้วยสมาชิกของเซต $S1$ หรือ $S2$ ใช้สัญลักษณ์ '`|`' ดังรูป 4.2



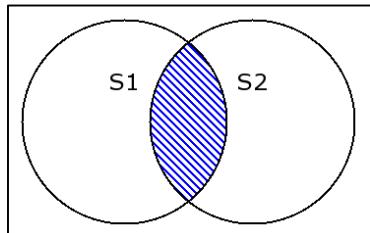
รูปที่ 4.2 $S1 | S2$

```
S1=set((1,1,2,3,5,8,13))
S2=set((2,3,5,7,11,13))
print("Set S1 :",S1)
print("Set S2 :",S2)
print("Set S1 | S2 :",S1|S2)
```



OUTPUT Set S1 : {1, 2, 3, 5, 8, 13}
Set S2 : {2, 3, 5, 7, 11, 13}
Set S1 | S2 : {1, 2, 3, 5, 7, 8, 11, 13}

อินเตอร์เซกชัน (Intersection) ของเซต S1 และ S2 คือ เซตที่ประกอบด้วยสมาชิกของเซต S1 และ S2 ใช้สัญลักษณ์ '`&`' ดังรูป 4.3



รูปที่ 4.3 S1 & S2

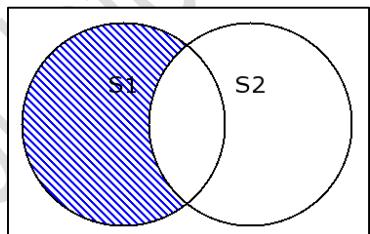
```
print("Set S1 :",S1)
print("Set S2 :",S2)
print("Set S1 & S2 :",S1&S2)
```



OUTPUT

```
Set S1 : {1, 2, 3, 5, 8, 13}
Set S2 : {2, 3, 5, 7, 11, 13}
Set S1 & S2 : {2, 3, 5, 13}
```

ผลต่าง (Difference) ของเซต S1 และ S2 คือเซตที่ประกอบด้วยสมาชิกที่เป็นสมาชิกของเซต S1 แต่ไม่เป็นสมาชิกของเซต S2 ใช้สัญลักษณ์ '`-`' ดังรูป 4.4



รูปที่ 4.4 S1 - S2

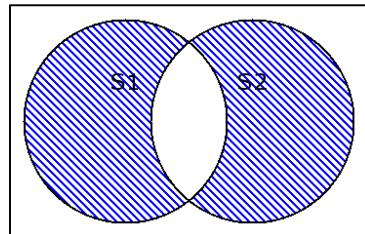
```
print("Set S1 :",S1)
print("Set S2 :",S2)
print("Set S1 - S2 :",S1-S2)
```



OUTPUT

```
Set S1 : {1, 2, 3, 5, 8, 13}
Set S2 : {2, 3, 5, 7, 11, 13}
Set S1 - S2 : {8, 1}
```

ผลต่างสมมาตร (Symmetric difference) ของเซต S1 และ S2 คือเซตที่ประกอบด้วยสมาชิกที่อยู่ในเซต S1 หรือเซต S2 แต่ไม่ใช่สมาชิกที่อยู่ทั้งใน S1 และ S2 ใช้สัญลักษณ์ '`^`' ดังรูป 4.5



รูปที่ 4.5 $S1 \wedge S2$

```
print("Set S1 :",S1)
print("Set S2 :",S2)
print("Set S1 ^ S2 :",S1^S2)
```



OUTPUT

```
Set S1 : {1, 2, 3, 5, 8, 13}
Set S2 : {2, 3, 5, 7, 11, 13}
Set S1 ^ S2 : {1, 7, 8, 11}
```

ตัวดำเนินการที่ใช้เปรียบเทียบสำหรับเซต (Set comparison operators)

สำหรับตัวดำเนินการที่ใช้กับเซตประกอบไปด้วย $<$, \leq , $>$, \geq , $=$, \neq , \in , not in ซึ่งจะอธิบายไว้อย่างละเอียดในหัวข้อนิพจน์ ตัวดำเนินการ และตัวถูกดำเนินการ แต่ในหัวข้อนี้จะอธิบายการใช้งานกับตัวดำเนินการพื้นฐาน เพื่อให้เข้าใจการทำงานของเซตในเบื้องต้น ตัวดำเนินการที่ใช้บ่อยๆ คือ \in และ not in ความหมายคือ เป็นสมาชิก หรือไม่เป็นสมาชิก ตัวอย่างเช่น

กำหนดให้ $S1 = \{1, 2, 3, 5, 8, 13\}$ และ $S2 = \{2, 3, 5, 7, 11, 13\}$

```
>>> S1 = {1, 2, 3, 5, 8, 13};
>>> S2 = {2, 3, 5, 7, 11, 13};
>>> S3 = {3, 5, 8}
>>> 3 in S1
True

>>> 7 in S1
False
```

เครื่องหมาย $<$, \leq , $>$, \geq ใช้สำหรับการเปรียบเทียบ 2 เซตใดๆ ถึงความเป็น superset และ subset เช่น ถ้า $S1$ เป็น subset ของ $S2$ ก็ต่อเมื่อสมาชิกทุกตัวใน $S1$ อยู่ใน $S2$ ใช้ตัวดำเนินการ \leq , $S1$ เป็น proper subset ของ $S2$ ใช้ตัวดำเนินการ $<$ เป็นตัน ดังตัวอย่างต่อไปนี้

```
>>> S1 < S2
False

S1 < S2 เป็นเท็จ เพราะสมาชิกใน S1 และ S2 มีค่าเท่ากัน
>>> S1 >= S3
True

S1 >= S3 เป็นจริง เพราะสมาชิกใน S1 มากกว่า S3
```

การสร้างเซตขึ้นสูง ผู้เขียนโปรแกรมสามารถสร้างเซตได้จากตัวแปรหลายชนิด เช่นเซตที่มีคุณสมบัติก่อไปนี้ ดังตัวอย่าง

```
>>> A = Set([('Yes', 'No'), ('Yes', 'Idontknow'), ('No',
   'Yes'), ('No', 'Idontknow')])
>>> ('Yes', 'No') in A
True
```

จากโปรแกรมตัวอย่างข้างบน เซต A ถูกสร้างมาจากตัวแปรชนิดทัพเพิล ('Yes', 'No'), ('Yes',
 'Idontknow'), ('No', 'Yes') และ ('No', 'Idontknow') ทั้งหมดถูกครอบด้วยลิสต์ [...] อีกครั้ง แล้วจึงทำ การแปลงจากลิสต์ไปเป็นตัวแปรชนิดเซต สังเกตเห็นว่าในบางครั้ง การสร้างตัวแปรเซตจากตัวแปรชนิด อื่นๆ จะให้ความสะดวกกว่าการกำหนดสมาชิกให้กับตัวแปรเซตตรงๆ

แม้ว่าเซตจะไม่ใช่ตัวแปรอันดับ (sequence) แต่ถ้าต้องการประมวลผลสมาชิกแต่ละตัวในเซต นิยมใช้ for (ซึ่งจะถูกอ่านว่า “สำหรับ” ในภาษาอังกฤษ เช่น “for” และการควบคุมทิศทางโปรแกรม) ในการเข้าถึง สมาชิกในเซต ดังตัวอย่างต่อไปนี้

```
even = set(range(2,10,2));
odd = set(range(1,10,2));
zero = set((0, '00'));
for n in even|odd|zero:
    print ("Member of set even|odd|zero :",n)
```



OUTPUT

```
Member of set even|odd|zero : 0
Member of set even|odd|zero : 1
Member of set even|odd|zero : 2
Member of set even|odd|zero : 3
Member of set even|odd|zero : 4
Member of set even|odd|zero : 5
Member of set even|odd|zero : 6
Member of set even|odd|zero : 7
Member of set even|odd|zero : 8
Member of set even|odd|zero : 9
Member of set even|odd|zero : 00
```

สรุปการใช้งานข้อมูลพื้นฐานและข้อมูลเชิงประกอบ

จากรายงานที่ 4.3 แสดงการเปรียบเทียบการใช้งานของตัวแปรชนิดพื้นฐานคือ ตริงกับข้อมูล เชิงประกอบคือ ลิสต์ ดิกชันนารี ทัพเพิลและเซ็ต ดังนี้

ตารางที่ 4.3 แสดงสรุปการใช้งานข้อมูลพื้นฐานและข้อมูลเชิงประกอบ

การดำเนินการ	ตริง	ลิสต์	ดิกชันนารี	ทัพเพิล	เซ็ต
การสร้าง ①	'...' หรือ "..." เช่น	[...] เช่น L = [1, 2,..., n]	{...} เช่น	(...) เช่น	{...} เช่น Z = {1, 2,..., n}

	$S = "HelloWorld"$		$D = \{key1:1,key2:2,\dots, keyn:n\}$	$T = (a, b,\dots,n)$	
การเข้าถึงสมาชิก ❷	$S[i]$ เช่น $S[0], S[-1]$	$L[i]$ เช่น $L[0], L[-1]$	$D[key]$ เช่น $D['key1']$	$T[i]$ เช่น $T[0], T[-1]$	for z in Z : เช่น for z in Z : print(z)
ตรวจสอบความ เป็นสมาชิก ❸	Char in S เช่น 'H' in S	e in L เช่น 1 in L	key in D เช่น 'key1' in D	e in T เช่น 2 in T	e in Z เช่น 1 in Z
การลบสมาชิก 1 ตัว ❹	ลบไม่ได้ แต่ใช้ การสร้างใหม่ได้ เช่น $S = S[:i-1]+ S[i:]$	del $L[i]$ เช่น del $L[0]$	del $D[key]$ เช่น del $D['key1']$	ลบไม่ได้ แต่ให้ แปลงเป็นลิสต์ ก่อนแล้วจึงลบ, การลบทัพเพิล ใช้ del T	$Z.remove(e)$ เช่น $Z.remove(1)$ หรือ $Z.pop()$
การแก้ไขสมาชิก ❺	แก้ไขไม่ได้ แต่ใช้ การสร้างใหม่ได้ เช่น $S = S[:i-1]+ new + S[i:]$	$L[i] = new$ เช่น $L[0] = 5$	$D[key] = new$ เช่น $D['key1'] = 5$	แก้ไขไม่ได้ แต่ ให้แปลงเป็น ลิสต์ก่อนแล้วจึง แก้ไข แล้วจึง แปลงกลับเป็น ทัพเพิลอีกรอบ	แก้ไขไม่ได้
เพิ่มสมาชิกใหม่ ❻	เพิ่มสมาชิกตรงๆ ไม่ได้ใช้ $S = S+ 'new'$ แทน	$L.append(e)$ เช่น $L.append(5)$	$D[newkey] = new$ เช่น $D['key5'] = 5$	เพิ่มไม่ได้ แต่ ให้แปลงเป็น ลิสต์ก่อนแล้วจึง เพิ่ม	$Z.add(e)$ เช่น $Z.add(5)$
การลบสมาชิก หลายตัว แบบต่อเนื่อง ❼	ลบไม่ได้ แต่ใช้ การสร้างใหม่ได้ เช่น $S = S[:i] +S[k:]$	del $L[i:k]$ เช่น del $L[2:4]$	ลบไม่ได้ แต่ถ้าเคลียร์ ข้อมูลทั้งหมด สามารถทำได้ คือ $D.clear()$	ลบไม่ได้ แต่ให้ แปลงเป็นลิสต์ ก่อนแล้วจึงลบ	ลบไม่ได้ แต่ถ้า เคลียร์ข้อมูลทั้ง หมดสามารถ ทำได้ คือ $Z.clear()$
แก้ไขสมาชิก หลายตัว แบบต่อเนื่อง ❽	แก้ไขไม่ได้ แต่ใช้ การสร้างใหม่ได้ เช่น $S = S[:i-1]+ news + S[i:]$	$L[i:k] =Lnews$ เช่น $L[2:4] = [0, 0]$	แก้ไขไม่ได้	แก้ไขไม่ได้ แต่ ให้แปลงเป็น ลิสต์ก่อนแล้วจึง แก้ไข	แก้ไขไม่ได้
การเพิ่มสมาชิก มากกว่า 1 ตัว ❾	เพิ่มไม่ได้ แต่ใช้ การสร้างใหม่ได้ เช่น $S = S +news$	$L.extend(Lnews)$ หรือ $L +L + Lnews$	$D.update(newD)$ เช่น $D.update({'key3':3})$	เพิ่มไม่ได้ แต่ ให้แปลงเป็น ลิสต์ก่อนแล้วจึง เพิ่ม	$Z.update(newZ)$ เช่น $Z.update({1, 2})$

❶ การสร้าง

ตัวแปรชนิดสตริง $\Rightarrow S = \text{'Hello World'}$ หรือ $S = \text{"Hello World"}$

ตัวแปรชนิดลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$, $L = [\text{'Yes'}, \text{'No'}, \text{'OK'}]$, $L = [1, 0.5, \text{'True'}$]

ตัวแปรชนิดติกซันนารี $\Rightarrow D = \{1:\text{'One'}, 1:\text{'Two'}, 1:\text{'Three'}\}$, $D = \{\text{'key1':1}, \text{'key2':2}\}$

ตัวแปรชนิดทัพเพิล $\Rightarrow T = (1, 2, 3, 4)$, $T = (1, 0.5, \text{'True'})$

ตัวแปรชนิดเซ็ต $\Rightarrow Z = \{1, 2, 3, 4\}$, $T = \{1, 0.5, \text{'True'}\}$

② การเข้าถึงสมาชิก

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $S[0] \Rightarrow \text{'H'}$, $S[-1] \Rightarrow \text{'d'}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $L[0] \Rightarrow 1$, $S[-1] \Rightarrow 5$

ติกซันนารี $\Rightarrow D = \{\text{'key1':1}, \text{'key2':2}\}$ และ $L[\text{'key1'}] \Rightarrow 1$

ทัพเพิล $\Rightarrow T = (1, 2, 3, 4)$ และ $T[0] \Rightarrow 1$, $S[-1] \Rightarrow 4$

เซ็ต $\Rightarrow Z = \{1, 2, 3, 4\}$ และ $\text{for } z \text{ in } Z: \text{print}(z) \Rightarrow \text{พิมพ์เลข 1 ถึง 4}$

③ การตรวจสอบความเป็นสมาชิก

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $\text{'H'} \text{ in } S \Rightarrow \text{True}$, $\text{'X'} \text{ in } S \Rightarrow \text{False}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $1 \text{ in } L \Rightarrow \text{True}$, $9 \text{ in } S \Rightarrow \text{False}$

ติกซันนารี $\Rightarrow D = \{\text{'key1':1}, \text{'key2':2}\}$ และ $\text{'key1'} \text{ in } D \Rightarrow \text{True}$

ทัพเพิล $\Rightarrow T = (1, 2, 3, 4)$ และ $1 \text{ in } T \Rightarrow \text{True}$, $9 \text{ in } T \Rightarrow \text{False}$

เซ็ต $\Rightarrow Z = \{1, 2, 3, 4\}$ และ $1 \text{ in } Z \Rightarrow \text{True}$, $9 \text{ in } Z \Rightarrow \text{False}$

④ การลบสมาชิก

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $S = S[:3-1] + S[6:] \Rightarrow \text{'HeWorld'}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $\text{del } L[0] \Rightarrow [2, 3, 4, 5]$

ติกซันนารี $\Rightarrow D = \{\text{'key1':1}, \text{'key2':2}\}$ และ $\text{del } D[\text{'key1'}] \Rightarrow \{\text{'key2':2}\}$

ทัพเพิล \Rightarrow ไม่สนับสนุนการลบสมาชิกต้องแบ่งเป็นลิสต์ก่อนแล้วจึงลบ

เซ็ต $\Rightarrow Z = \{1, 2, 3, 4\}$ และ $Z.\text{remove}(1) \Rightarrow \{2, 3, 4\}$

⑤ การแก้ไขหรือปรับปรุงสมาชิก

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $S = S[:3-1] + \text{'XX'} + S[6:] \Rightarrow \text{'HeXXWorld'}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $L[0] = 0 \Rightarrow [0, 2, 3, 4, 5]$

ดิกชันนารี $\Rightarrow D = \{\text{'key1':1}, \text{'key2':2}\}$ และ $D[\text{'key1'}] = 9 \Rightarrow \{\text{'key1':9}, \text{'key2':2}\}$

ทัพเพิล \Rightarrow ไม่สนับสนุนการแก้ไขสมาชิกต้องแปลงเป็นลิสต์ก่อนแล้วจึงแก้ไข

เช็ต \Rightarrow ไม่สนับสนุนการแก้ไขสมาชิก

⑥ การเพิ่มสมาชิกใหม่

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $S = \text{'X'} + S + \text{'X'} \Rightarrow \text{'XHello WorldX'}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $L.append(6) \Rightarrow [1, 2, 3, 4, 5, 6]$

ดิกชันนารี $\Rightarrow D = \{\text{'key1':1}, \text{'key2':2}\}$ และ $D[\text{'key3'}] = 3 \Rightarrow \{\text{'key1':1}, \text{'key2':2}, \text{'key3':3}\}$

ทัพเพิล \Rightarrow ไม่สนับสนุนการเพิ่มสมาชิกใหม่ต้องแปลงเป็นลิสต์ก่อนแล้วจึงเพิ่ม

เช็ต $\Rightarrow Z = \{1, 2, 3, 4\}$ และ $Z.add(5) \Rightarrow \{1, 2, 3, 4, 5\}$

⑦ การลบสมาชิกหลายตัวแบบต่อเนื่อง

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $S = S[:2] + S[7:] \Rightarrow \text{'Heorld'}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $del L[2:4] \Rightarrow [1, 2, 5]$

ดิกชันนารี $\Rightarrow D = \{\text{'key1':1}, \text{'key2':2}\}$ และ $D.clear() \Rightarrow \{\}$

ทัพเพิล \Rightarrow ไม่สนับสนุนการลบสมาชิกใหม่ต้องแปลงเป็นลิสต์ก่อนแล้วจึงลบ

เช็ต $\Rightarrow Z = \{1, 2, 3, 4\}$ และ $Z.clear() \Rightarrow set()$

⑧ การแก้ไขหรือปรับปรุงสมาชิกหลายตัวแบบต่อเนื่อง

สตริง $\Rightarrow S = \text{'Hello World'}$ และ $S = S[:3-1] + \text{'XX'} + S[6:] \Rightarrow \text{'HeXXWorld'}$

ลิสต์ $\Rightarrow L = [1, 2, 3, 4, 5]$ และ $L[2:4] = [0, 0] \Rightarrow [1, 2, 0, 0, 5]$

ดิกชันนารี \Rightarrow ไม่สนับสนุนการแก้ไข

ทัพเพิล \Rightarrow ไม่สนับสนุนการแก้ไขสมาชิกต้องแปลงเป็นลิสต์ก่อนแล้วจึงแก้ไข

เช็ต \Rightarrow ไม่สนับสนุนการแก้ไข

๙ การเพิ่มสมาชิกหลายตัวแบบต่อเนื่อง

สตริง \Rightarrow $S = \text{'Hello World'}$ และ $S = S + \text{' Python'}$ \Rightarrow $\text{'Hello World Python'}$

ลิสต์ \Rightarrow $L = [1, 2, 3, 4, 5]$ และ $L.extend([6, 7]) = [0, 0] \Rightarrow [1, 2, 3, 4, 5, 6, 7]$

ดิกชันนารี \Rightarrow $D = \{\text{'key1':1, 'key2':2}\}$ และ $D.update(\{\text{'key3':3, 'key4':4}\}) \Rightarrow \{\text{'key1': 1, 'key2': 2, 'key3': 3, 'key4': 4}\}$

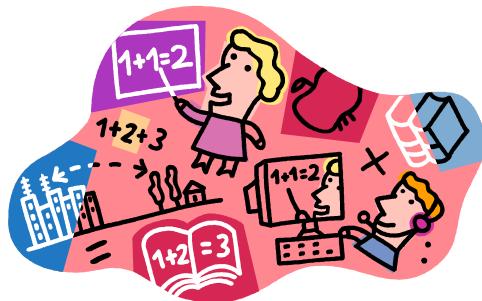
ทัพเพิล \Rightarrow ไม่สนับสนุนการเพิ่มสมาชิกต้องแปลงเป็นลิสต์ก่อนแล้วจึงเพิ่ม

เช็ต \Rightarrow $Z = \{1, 2, 3, 4\}$ และ $Z.update(\{5, 6\}) \Rightarrow \{1, 2, 3, 4, 5, 6\}$

จบบทที่ 4

บทที่ 5

นิพจน์ ตัวดำเนินการ และตัวถูกดำเนินการ (Expression, Operators and Operands)



นิพจน์ (Expression) คือ การดำเนินการที่ประกอบด้วยตัวดำเนินการ (Operator) และตัวถูกดำเนินการ (Operand)

ตัวดำเนินการ (Operator) คือ สัญลักษณ์ที่ใช้ในการแทนการกระทำอย่างใดอย่างหนึ่งกับข้อมูล เช่น `+`, `-`, `*`, `**`, `/`, `//`, `%`, `=`, `>`, `<`, `!=`, `==`, `<>` เป็นต้น

ตัวถูกดำเนินการ (Operand) คือ ข้อมูลที่ถูกกระทำโดยตัวดำเนินการ ซึ่งตัวถูกดำเนินการอาจอยู่ในรูปของตัวแปร (Variable) ค่าคงที่ (Constant) ค่าที่ได้รับจากฟังก์ชัน (Return function) หรือแม้กระทั่งนิพจน์ (Expression) เอง ก็เป็นตัวถูกดำเนินการได้ เช่น `a+b`, `x++`, `a != b`, `x = (a + b) / 2` เป็นต้น จากตัวอย่าง $y = x^2 + 2x + 1$

`y`, `x`, `1` คือ ตัวถูกดำเนินการ (Operand)

`=`, `+`, `ยกกำลังสอง` คือ ตัวดำเนินการ (Operator)

$y = x^2 + 2x + 1$ ทั้งหมด คือ นิพจน์ (Expression)

ภาษา Python สนับสนุนตัวดำเนินการหลายแบบ คือ Arithmetic Operators, Comparison Operators, Assignment Operators, Logical Operators, Bitwise Operators, Membership Operators, Identity Operators

1. ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

เพื่อให้เข้าใจถึงการทำงานของตัวดำเนินการทางคณิตศาสตร์ สมมุติให้ ตัวแปร `a = 10`, `b = 5`, `c = 9.0`, `d = 2.0`, `e = -3.5` สำหรับตัวอย่างการดำเนินการทางคณิตศาสตร์แสดงในตารางที่ 4.1

ตารางที่ 4.1 Arithmetic Operators

Operator	ความหมาย	ตัวอย่าง ($a = 10$, $b = 5$, $c = 9.0$, $d = 2.0$, $e = -3.5$)
+	การบวก (Addition)	$a + b = 15$, $a + c = 19.0$
-	การลบ (Subtraction)	$a - b = 5$, $b - a = -5$, $c - d = 7.0$, $d - c = -7.0$
*	การคูณ (Multiplication)	$a * b = 50$, $b * d = 10.0$, $a * e = -35.0$, $e * e = 12.25$
/	การหาร (Division)	$a / b = 2.0$, $b / a = 0.5$, $d / c = 2.99999$, $c / b = 1.8$, $e / d = -1.75$, $e / e = 1.0$
%	การหารเอาเศษ (Modulus)	$\begin{array}{r} 2 \\ 5 \overline{)10} \\ \underline{-10} \\ 0 \end{array}$ $a \% b = 0 \quad \text{หรือ } b \% a = 5$ $\begin{array}{r} 0 \\ 10 \overline{)5} \\ \underline{-5} \\ 0 \end{array}$ $c \% d = 1.0, c \% e = -1.5, e \% c = 5.5$
**	เลขยกกำลัง (Exponent)	$a^{**}b (a^b) = 100000$ หรือ $b^{**}a (b^a) = 9,765,625$ $d^{**}e = 0.0883$, $e^{**}d = -12.25$
//	การหารเอาส่วน (Floor division)	$\begin{array}{r} 2 \\ 5 \overline{)10} \\ \underline{-10} \\ 0 \end{array}$ $a // b = 2 \quad \text{หรือ } b // a = 0$ <p>ถ้า $9 // 2 = 4$, $9.0 // 2.0 = 4.0$, $9.0 // 2 = 4.0$ และ $9 // 2.0 = 4.0$</p>

ตัวอย่าง Arithmetic Operators

```

a = 21
b = 10
c = 0
c = a + b
print ("1: Value of c is ", c)
c = a - b
print ("2: Value of c is ", c)
c = a * b
print ("3: Value of c is ", c)
c = a / b
print ("4: Value of c is ", c)
c = a % b
print ("5: Value of c is ", c)
a = 2
b = 3
c = a**b
print ("6: Value of c is ", c)
a = 10
b = 5
c = a//b
print ("7: Value of c is ", c)

```


OUTPUT

```

1: Value of c is 31
2: Value of c is 11
3: Value of c is 210
4: Value of c is 2.1
5: Value of c is 1
6: Value of c is 8
7: Value of c is 2

```

2. ตัวดำเนินการทางด้านการเปรียบเทียบ (Comparison Operators)

เป็นการเปรียบเทียบกันระหว่างข้อมูล หรือตัวถูกดำเนินการอย่างน้อย 2 จำนวน โดยการเปรียบเทียบนั้นจะทำการเปรียบเทียบที่ลีบ 2 จำนวน ผลลัพธ์ที่ได้จะอยู่ในรูปตรรกะ (Boolean) คือ จริง (True) หรือ เท็จ (False) ถ้ามีการเปรียบเทียบมากกว่า 2 จำนวน จะทำการเปรียบเทียบจากด้านซ้ายมือไปขวา มือ หรือภายในวงเล็บก่อน โดยมีการเชื่อมด้วยตรรกะ เช่น and (และ) or (หรือ) ดังตัวอย่างในตารางที่ 4.2 โดยสมมุติว่า ตัวแปร $a = 10$, $b = 5$, $c = 9.0$, $d = 2.0$, $e = -3.5$

ตารางที่ 4.2 Comparison Operators

Operator	ความหมาย	ตัวอย่าง ($a = 10$, $b = 5$, $c = 9.0$, $d = 2.0$, $e = -3.5$)
$==$	เท่ากับ (Equal)	เป็นจริงก็ต่อเมื่อ ซ้ายเท่ากับขวา เช่น $a == b$ ให้ผลลัพธ์ คือ เท็จ $x = 'str'; y = 'stR'; x == y$ ผลลัพธ์คือ เท็จ
$!=$	ไม่เท่ากับ (Not equal)	เป็นจริงก็ต่อเมื่อ ซ้ายไม่เท่ากับขวา เช่น $b != c$ ให้ผลลัพธ์ คือ จริง
$>$	มากกว่า(Greater than)	เป็นจริงก็ต่อเมื่อ ซ้ายมากกว่าขวา เช่น $c > d$ ให้ผลลัพธ์ คือ จริง
$<$	น้อยกว่า (Less than)	เป็นจริงก็ต่อเมื่อ ซ้ายน้อยกว่าขวา เช่น $d < e$ ให้ผลลัพธ์ คือ เท็จ
$>=$	มากกว่าหรือเท่ากับ (Greater than or equal to)	เป็นจริงก็ต่อเมื่อ ซ้ายมากกว่าหรือเท่ากับขวา เช่น $a >= c$ ให้ผลลัพธ์คือ จริง, $(3 >= 5)$ ให้ผลลัพธ์คือ เท็จ
$<=$	น้อยกว่าหรือเท่ากับ (Less than or equal to)	เป็นจริงก็ต่อเมื่อ ซ้ายน้อยกว่าหรือเท่ากับขวา เช่น $b <= c$ ให้ผลลัพธ์คือ จริง, $(5.0 <= 5)$ ให้ผลลัพธ์คือ จริง



Note: ภาษาไพธอนไม่สนับสนุนตัวดำเนินการ $==$, $!=$, $<$, $>$ เหมือนในภาษา C/C++

ตัวอย่าง Comparison Operators

$a = 21$

$b = 10$

```

c = 0
if ( a == b ):
    print ("Line 1 - a is equal to b")
else:
    print ("Line 1 - a is not equal to b")
if ( a != b ):
    print ("Line 2 - a is not equal to b")
else:
    print ("Line 2 - a is equal to b")
if ( a < b ):
    print ("Line 3 - a is less than b")
else:
    print ("Line 3 - a is not less than b")
if ( a > b ):
    print ("Line 4 - a is greater than b")
else:
    print ("Line 4 - a is not greater than b")

a = 5;
b = 20;
if ( a <= b ):
    print("Line 5 - a is either less than or equal to b")
else:
    print("Line 5 - a is neither less than nor equal to b")
if ( b >= a ):
    print("Line 6 - b is either greater than or equal to b")
else:
    print("Line 6 - b is neither greater than nor equal to b")

```


OUTPUT

```

Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not less than b
Line 4 - a is greater than b
Line 5 - a is either less than or equal to b
Line 6 - b is either greater than or equal to b

```

3. ตัวดำเนินการกำหนดค่า (Assignment Operators)

เป็นสัญลักษณ์ที่ใช้สำหรับกำหนดค่า หรือเปลี่ยนแปลงค่าให้แก่ตัวแปร ซึ่งตัวแปรทางด้านซ้ายมือจะถูกกำหนดค่าจากข้อมูลหรือตัวแปรจากด้านขวา มีอยู่ 5 ชนิด คือ $a = 10, b = 5, c = 9.0, d = 2.0, e = -3.5$

ตารางที่ 4.3 Assignment Operators

Operator	ความหมาย	ตัวอย่าง ($a = 10, b = 5, c = 9.0, d = 2.0, e = -3.5$)
----------	----------	--

=	กำหนดค่า (Simple assignment operator)	กำหนดค่าทางด้านขวาเมื่อ (ข้ามมือของผู้อ่าน) ให้กับตัว变量ทางด้านซ้ายเมื่อ เช่น $a = b + c$ (a มีค่าเท่ากับ 14.0), $d = 4.0$ (d มีค่าเท่ากับ 4.0)
+=	บวกก่อนกำหนดค่า (Add and assignment operator)	นำค่าทางซ้ายมือบวกทางขวาแล้วเก็บไว้ในตัว变量ทางซ้ายเมื่อ เช่น $a += b$ ($a = a + b$) ซึ่ง a มีค่าเท่ากับ 15
-=	ลบก่อนกำหนดค่า (Subtract and assignment operator)	นำค่าทางซ้ายมือลบทางขวาแล้วเก็บไว้ในตัวvariablesทางซ้ายเมื่อ เช่น $a -= b$ ($a = a - b$) ซึ่ง a มีค่าเท่ากับ 5
*=	คูณก่อนกำหนดค่า (Multiply and assignment operator)	นำค่าทางซ้ายคูณทางขวาแล้วเก็บไว้ในตัวvariablesทางซ้ายเมื่อ เช่น $a *= b$ ($a = a * b$) ซึ่ง a มีค่าเท่ากับ 50
/=	หารก่อนกำหนดค่า (Divide and assignment operator)	นำค่าทางซ้ายหารทางขวาแล้วเก็บไว้ในตัวvariablesทางซ้ายเมื่อ เช่น $a /= b$ ($a = a / b$) ซึ่ง a มีค่าเท่ากับ 2.0
%=	หารเอาเศษก่อนกำหนดค่า (Modulus and assignment operator)	นำค่าซ้ายหารแบบเอาเศษด้วยค่าทางขวาเมื่อ และเก็บเศษที่ได้ไว้ทางซ้ายเมื่อ เช่น $a %= b$ ($a = a \% b$) ซึ่ง a มีค่าเท่ากับ 0
**=	ยกกำลังก่อนกำหนดค่า (Exponent and assignment operator)	นำค่าซ้ายยกกำลังด้วยค่าทางขวาเมื่อ และค่าที่ได้ไว้ทางซ้ายเมื่อ เช่น $a **= b$ ($a = a ** b$) ซึ่ง a มีค่าเท่ากับ 10,000
//=	หารเอาส่วนก่อนกำหนดค่า (Floor division and assigns a value)	นำค่าซ้ายหารเอาส่วนด้วยค่าทางขวาเมื่อ และค่าที่ได้ไว้ทางซ้ายเมื่อ เช่น $a //= b$ ($a = a // b$) ซึ่ง a มีค่าเท่ากับ 2

ตัวอย่าง Assignment Operators

```

a = 21
b = 10
c = 0
c = a + b
print ("Line 1 - Value of c is ", c)
c += a
print ("Line 2 - Value of c is ", c)
c *= a
print ("Line 3 - Value of c is ", c)
c /= a
print ("Line 4 - Value of c is ", c)

```

```

c = 2
c %= a
print ("Line 5 - Value of c is ", c)
c **= a
print ("Line 6 - Value of c is ", c)
c /= a
print ("Line 7 - Value of c is ", c)

```


OUTPUT

```

Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52.0
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864

```

4. ตัวดำเนินการระดับบิต (Bitwise Operators)

ก่อนทำความเข้าใจเกี่ยวกับตัวดำเนินการระดับบิต จำเป็นต้องเข้าใจเกี่ยวกับตารางความจริง (Truth table) ของตัวดำเนินการ AND (และ), OR (หรือ), NOT (ไม่), XOR ก่อน ในตารางที่ 4.4

ตารางที่ 4.4 Truth table

ค่าในตัวแปร		ตัวดำเนินการ			
A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

จากตารางที่ 4.4 มีข้อสังเกตว่า

AND ผลลัพธ์มีค่าเท่ากับ จริง (1) ก็ต่อเมื่อ A และ B ต้องเป็นจริงทั้งคู่

OR ผลลัพธ์มีค่าเท่ากับ เท็จ (0) ก็ต่อเมื่อ A และ B ต้องเป็นเท็จทั้งคู่

XOR ผลลัพธ์มีค่าเท่ากับ จริง (1) ก็ต่อเมื่อ ค่า A และ B เหมือนกัน เช่น A, B = 0 หรือ A, B = 1 และ ผลลัพธ์มีค่าเท่ากับ เท็จ (0) ก็ต่อเมื่อ ค่า A และ B ต่างกัน เช่น A = 0, B = 1 หรือ A = 1, B = 0 เป็นต้น



Tips: การหาค่า AND, OR, XOR ให้ได้รวดเร็วคือ กรณี AND ถ้าค่าได้ค่าหนึ่งเป็น 0 จะทำให้ผลลัพธ์เป็นเท็จทันที กรณี OR ถ้าค่าได้ค่าหนึ่งเป็น 1 จะทำให้ผลลัพธ์เป็นจริงทันที และ XOR คือเหมือนกันได้ 0 ต่างกันได้ 1



Note: ภาษา Python ถือว่าค่าต่างๆ เหล่านี้เป็นเท็จ (0) คือ False, 0, None, สริงที่มีค่าว่าง "", ข้อมูลชนิด list ที่มีค่าเป็น [], ตัวแปร tuples ที่ว่าง ()

ตัวดำเนินการระดับบิต คือ ตัวดำเนินการที่กระทำกับตัวถูกกระทำในระดับบิตต่อบิต คล้ายภาษาในระดับต่ำ ซึ่งทำให้โปรแกรมสามารถทำงานได้เร็วขึ้น ซึ่งมี 6 ตัว ดังตารางที่ 4.5 สมมุติให้ $a = 60$ และ $b = 13$ เพื่อแสดงให้เห็นการทำงานของตัวดำเนินการระดับบิต จำเป็นต้องแสดงให้อยู่ในเลขฐานสอง คือ

$$a \text{ (60 ฐานสิบ)} = 0011\ 1100 \text{ (ฐานสอง)} \text{ และ } b = 0000\ 1101$$

ตัวอย่างการใช้ Bitwise

ตัวดำเนินการชนิด and $\Rightarrow a \& b = 0000\ 1100$

$$\begin{array}{r} 0011\ 1100 \\ 0000\ 1101 \\ \hline \underline{0000\ 1100} \end{array} \text{ and}$$

ตัวดำเนินการชนิด or $\Rightarrow a | b = 0011\ 1101$

$$\begin{array}{r} 0011\ 1100 \\ 0000\ 1101 \\ \hline \underline{0011\ 1101} \end{array} \text{ or}$$

ตัวดำเนินการชนิด xor $\Rightarrow a ^ b = 0011\ 0001$

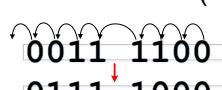
$$\begin{array}{r} 0011\ 1100 \\ 0000\ 1101 \\ \hline \underline{0011\ 0001} \end{array} \text{ xor}$$

ตัวดำเนินการชนิด not $\Rightarrow \sim a = 1100\ 0011$

$$\begin{array}{r} 0011\ 1100 \\ \hline \underline{1100\ 0011} \end{array} \text{ not}$$

ตารางที่ 4.5 Bitwise Operators

Operator	ความหมาย	ตัวอย่าง ($a = 60$ และ $b = 13$)
&	and ระดับบิต (Binary AND)	เป็นจริงก็ต่อเมื่อซ้ายและขวาเป็นจริง เช่น $a \& b$ หรือ a and b ผลลัพธ์คือ 0000 1100 (เท่ากับ 12 ฐานสิบ)
	or ระดับบิต (Binary OR)	เป็นจริงก็ต่อเมื่อซ้ายหรือขวาเป็นจริง เช่น $a b$ หรือ a or b ผลลัพธ์คือ 0011 1101 (เท่ากับ 60)

<code>^</code>	XOR ระดับบิต (Binary XOR)	เป็นจริงก็ต่อเมื่อค่าด้านซ้ายหรือขวามีค่าต่างกัน $a ^ b$ ผลลัพธ์คือ 0011 0001 (เท่ากับ 49)
<code>~</code>	คอมพลีเมนต์ (Binary Ones Complement)	ทำการสลับบิตข้อมูลโดยใช้หลักการของ 2's 补 $(\sim a)$ ผลลัพธ์คือ -61
<code><<</code>	เลื่อนบิตข้อมูลไปทางซ้าย (Binary Left Shift)	ทำการเลื่อนบิตข้อมูลไปทางซ้ายมีครั้งละ n บิต เช่น $a << 1$ จะเลื่อนบิตไปทางซ้าย 1 บิต ผลลัพธ์คือ 0111 1000 (120) ถ้าต้องการเลื่อนทีละ 2 บิต ทำได้โดย $a << 2 = 1111 0000$ (240) 
<code>>></code>	เลื่อนบิตข้อมูลไปทางขวา (Binary Right Shift)	ทำการเลื่อนบิตข้อมูลไปทางขวามีครั้งละ n บิต เช่น $a >> 1$ จะเลื่อนบิตไปทางขวา 1 บิต ผลลัพธ์คือ 0001 1110 (30) ถ้าต้องการเลื่อนทีละ 2 บิต ทำได้โดย $a >> 2 = 0000 1111$ (15) 



Tips: เมื่อทำการเลื่อนบิตไปทางซ้าย `<<` คือการคูณ และเลื่อนบิตไปทางขวา `>>` คือการหาร

ตัวอย่างการใช้ Bitwise

```

a = 60      # 60 = 0011 1100
b = 13      # 13 = 0000 1101
c = 0
c = a & b    # 12 = 0000 1100
print ("Line 1 - Value of c is ", c)
c = a | b    # 61 = 0011 1101
print ("Line 2 - Value of c is ", c)
c = a ^ b    # 49 = 0011 0001
print ("Line 3 - Value of c is ", c)
c = ~a       # -61 = 1100 0011
print ("Line 4 - Value of c is ", c)
c = a << 2   # 240 = 1111 0000
print ("Line 5 - Value of c is ", c)
c = a >> 2   # 15 = 0000 1111
print ("Line 6 - Value of c is ", c)

```



OUTPUT

```

Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240

```

Line 6 - Value of c is 15

5. ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)

ตัวดำเนินการทางตรรกศาสตร์ คือ ตัวดำเนินการที่ใช้ในการเปรียบเทียบ และตัดสินใจ โดยมีเงื่อนไขดังต่อไปนี้ นำมาเปรียบเทียบกัน ผลที่ได้จากการเปรียบจะได้ผลเป็น 2 กรณี คือ จริงซึ่งให้ค่าเป็น 1 และเท็จซึ่งให้ค่าเป็น 0 ในภาษาไพธอนมี 3 ตัวคือ and (และ) or (หรือ) not (ไม่) ดังในตารางที่ 4.6 โดยกำหนดให้ $a = 10$, $b = 20$, $c=0$

ตารางที่ 4.6 Logical Operators

Operator	ความหมาย	ตัวอย่าง ($a = 10$, $b = 20$, $c=0$)
and	และ (and) (Logical AND)	เงื่อนไขจะเป็นจริง เมื่อค่าข้อมูลด้านซ้ายและขวาเป็นจริงทั้งคู่ เช่น $a \text{ and } b = \text{จริง}$, $a \text{ and } c = \text{เท็จ}$
or	or (หรือ) (Logical OR)	เงื่อนไขจะเป็นจริง เมื่อค่าข้อมูลด้านซ้ายหรือขวาเป็นจริง เช่น $a \text{ or } b = \text{จริง}$, $c \text{ or } c = \text{เท็จ}$
not	ไม่ (Logical NOT)	ทำการเปลี่ยนค่าเป็นตรงกันข้าม ถ้าค่าเป็น 0 เมื่อใช้ตัวดำเนินการ NOT จะทำให้ค่าที่ได้เป็น 1 เช่น $\text{not } (a \text{ and } b) = \text{False}$, $\text{not } 0 = \text{True}$, $\text{not } 1 = \text{False}$, $\text{not } \text{False} = \text{True}$, $\text{not } \text{True} = \text{False}$

ตัวอย่างการใช้ Logic Operations

```

a = 10
b = 20
c = 0

if (a and b):
    print("Line 1 - a and b are true")
else:
    print("Line 1 - Either a is not true or b is not true")
if (a or b):
    print("Line 2 - Either a is true or b is true or both are true")
else:
    print("Line 2 - Neither a is true nor b is true")

a = 0
if (a and b):
    print("Line 3 - a and b are true")
else:
    print("Line 3 - Either a is not true or b is not true")
if (a or b):
    print("Line 4 - Either a is true or b is true or both are true")
else:

```

```

print("Line 4 - Neither a is true nor b is true")
if not(a and b):
    print("Line 5 - a and b are true")
else:
    print("Line 5 - Either a is not true or b is not true")

```

OUTPUT

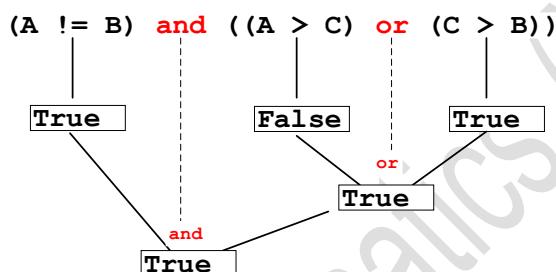
```

Line 1 - a and b are true
Line 2 - Either a is true or b is true or both are true
Line 3 - Either a is not true or b is not true
Line 4 - Either a is true or b is true or both are true
Line 5 - a and b are true

```

ตัวอย่างการหาผลลัพธ์ทางตรรก กำหนดให้ $A = 1, B = 2, C = 3$

ถ้า $(A \neq B) \text{ and } ((A > C) \text{ or } (C > B))$ ผลลัพธ์เป็นจริงหรือเท็จแสดงได้ดังนี้



จากรูปผลลัพธ์ที่ได้มีค่าเป็นจริง

6. ตัวดำเนินงานการเป็นสมาชิก (Membership Operators)

ไฟชอนสร้างตัวดำเนินการสมาชิกเพิ่มขึ้น 2 ตัว คือ `in` และ `not in` เพื่อใช้การทดสอบว่าอยู่ในเซตของข้อมูลหรือไม่ ตัวอย่างข้อมูลที่ตรวจสอบ เช่น ข้อมูลสายสตริง (strings) ลิสต์ (lists) ทับเบล (tuples) แสดงในตารางที่ 4.7 และกำหนดให้ `list = [1, 2, 3, 4, 5], a = 2, b = 7`

ตารางที่ 4.7 Membership operators

Operator	ความหมาย	ตัวอย่าง (<code>list = [1, 2, 3, 4, 5], a = 2, b = 7</code>)
<code>in</code>	อยู่ใน (in)	เป็นจริง เมื่อสามารถค้นหาข้อมูลพบ แต่ถ้าไม่พบจะให้ผลลัพธ์เป็นเท็จ เช่น <code>a in list</code> = จริง (True), <code>9 in list</code> = เท็จ (False), <code>b in list</code> = เท็จ
<code>not in</code>	ไม่อยู่ใน (not in)	เป็นจริง เมื่อไม่สามารถค้นหาข้อมูลพบ แต่ถ้าพบจะให้ผลลัพธ์เป็นจริง เช่น <code>a not in list</code> = จริง (True), <code>9 not in list</code> = เท็จ (False)

ตัวอย่างการใช้ Membership Operators

```

a = 10
b = 20
list = [1, 2, 3, 4, 5]
if (a in list):
    print("Line 1 - a is available in the given list")
else:
    print("Line 1 - a is not available in the given list")
if (b not in list):
    print("Line 2 - b is not available in the given list")
else:
    print("Line 2 - b is available in the given list")

a = 2
if (a in list):
    print("Line 3 - a is available in the given list")
else:
    print("Line 3 - a is not available in the given list")

```



OUTPUT

```

Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list

```

7. ตัวดำเนินการเอกลักษณ์ (Identity Operators)

ตัวดำเนินการเอกลักษณ์ใช้สำหรับเปรียบเทียบอ็อปเจกต์ 2 อ็อปเจกต์ใดๆ ที่เก็บในหน่วยความจำ มีอยู่ 2 ตัวคือ `is` และ `is not` ซึ่งคำสั่งดังกล่าวนำพาดแทนคำสั่ง `equal` เพราะ Python มองว่าคำสั่ง `equal` ไม่สื่อความหมาย ดังแสดงในตารางที่ 4.8

ตารางที่ 4.8 Identity operators

Operator	ความหมาย	ตัวอย่าง
<code>is</code>	เป็น, อยู่, คือ	เป็นจริง เมื่ออ็อปเจกต์ทั้งสองข้างเท่ากัน หรือเหมือนกันทุกประการ เช่น <code>10 is 10 = True</code> , <code>10 is 20 = False</code>
<code>is not</code>	ไม่เป็น, ไม่ใช่	เป็นจริง เมื่ออ็อปเจกต์ทั้งสองข้างไม่เท่ากัน หรือไม่เหมือนกันทุกประการ เช่น <code>10 is not 10 = False</code> , <code>10 is not 20 = True</code>

ตัวอย่าง Identity Operators

```

a = 20
b = 20
if (a is b):
    print("Line 1 - a and b have same identity")
else:
    print("Line 1 - a and b do not have same identity")
if (id(a) == id(b)):
    print("Line 2 - a and b have same identity")

```

```

else:
    print("Line 2 - a and b do not have same identity")

b = 30
if (a is b):
    print("Line 3 - a and b have same identity")
else:
    print("Line 3 - a and b do not have same identity")
if ( a is not b ):
    print("Line 4 - a and b do not have same identity")
else:
    print("Line 4 - a and b have same identity")

```



OUTPUT

```

Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity

```

8. ลำดับความสำคัญของตัวดำเนินการ (Operators Precedence)

Operators Precedence คือการเรียงลำดับการดำเนินการทางเครื่องหมายทางคณิตศาสตร์ ว่า จะให้ความสำคัญกับเครื่องหมายใดก่อนหรือหลัง โดยจะเรียงลำดับความสำคัญดังแสดงในตาราง 4.9 โดยลำดับที่มีนัยสำคัญสูงสุดจะอยู่ด้านบนของตาราง ต่ำสุดอยู่ด้านล่าง (ท้าย) ตาราง เช่น (...) มีนัยสำคัญสูงสุด และ not หรือ or มีนัยสำคัญต่ำสุด ตัวอย่างเช่น $x = 7 + 3 * 2$ ค่าที่ถูกต้องคือ x ต้องเท่ากับ 13 ไม่ใช่ 20 เนื่องจาก * มีลำดับความสำคัญมากกว่า + ดังนั้น จะต้องทำการคูณระหว่าง 3 และ 2 ให้เสร็จก่อน หลังจากนั้นจึงบวกด้วย 7 จึงเป็นคำตอบที่ถูกต้อง (ถ้านำ 7 + 3 ก่อนจากนั้นจึงคูณด้วย 2 จะเป็นคำตอบที่ผิด) ถ้าไม่แน่ใจกับลำดับของการประมวลผล ให้ใช้เครื่องหมาย (...) ช่วย เช่น $x = 7 + (3 * 2)$ เป็นต้น

ตารางที่ 4.9 Operators Precedence

Operator	คำอธิบาย
(...)	วงเล็บมีนัยสำคัญสูงสุด ต้องถูกกระทำก่อนเสมอ
**	ยกกำลัง
~, +, -	คอมพิลิเมนต์ การบวกก่อน และการลบก่อน เช่น $a += b$
*, /, %, //	การคูณ หาร มอต และการหารเอาส่วน
+, -	การบวก และการลบ
>>, <<	การเลื่อนขวา (shift right) และเลื่อนทางซ้าย (shift left)
&	การ and ระดับบิต
^,	XOR และ OR
<=, <, >, >=	น้อยกว่าหรือเท่ากับ น้อยกว่า มากกว่า และมากกว่าหรือเท่ากับ

<code>==, !=</code>	เท่ากัน และไม่เท่ากัน
<code>=, %=, /=, //=, -=, +=, *=, **=</code>	การกำหนดค่า หารເອາເສີ່ງ ພວກເຮົາສ່ວນ ລບກ່ອນກຳທັດຄ່າ ບວກກ່ອນ ກຳທັດຄ່າ ອຸນກ່ອນກຳທັດຄ່າ ຍກກຳລັງກ່ອນກຳທັດຄ່າ
<code>is, is not</code>	ຕັ້ງດຳເນີນການເອກລັກຂະໜົນ
<code>In, not in</code>	ຕັ້ງດຳເນີນການສມາຝຶກ
<code>not, or, and</code>	ไม่ (not), หรือ (or), และ (and)



Tips: จากสมการ $x = 7 + 3 * 2$ ถ้าผู้เขียนโปรแกรมมั่นใจอย่างแน่นอนว่า ต้องทำการบวก 7 และ 3 ก่อน แล้วนำผลลัพธ์ที่ได้ไปคูณกับ 2 ให้ใช้ (...) ครอบนิพจน์ที่ต้องการให้กระทำการบวก เช่น $x = (7 + 3) * 2$ ผลลัพธ์ที่ได้คือ 20



Note: เมื่อຕັ້ງດຳເນີນການມື້ນຍສຳຄັນເທົກນັນ เช่น * กັບ / ໄພຮອນຈະປະມວລຜລຈາກຫ້າຍໄປໆຂວາ
ເສນວ ເຊັ່ນ $5 * 6 / 4 \Rightarrow (5 * 6) / 4$ ປຸລັພົບທີ່ໄດ້ຄືວ່າ 7.5

ตัวอย่าง Operators Precedence

```
a = 20
b = 10
c = 15
d = 5
e = 0
e = (a + b) * c / d      # (30 * 15)/5
print ("Value of (a + b) * c / d is ", e)

e = ((a + b) * c) / d    # (30 * 15 )/5
print ("Value of ((a + b) * c) / d is ", e)

e = (a + b) * (c / d);   # (30) * (15/5)
print ("Value of (a + b) * (c / d) is ", e)

e = a + (b * c) / d;     # 20 + (150/5)
print ("Value of a + (b * c) / d is ", e)

e = a + b ** d - c;      # 100000 + 20 - 15
print ("Value of a + b ** d - c is ", e)
```



OUTPUT

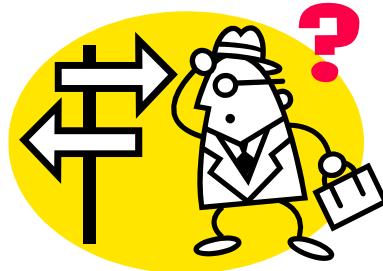
```
Value of (a + b) * c / d is 90.0
Value of ((a + b) * c) / d is 90.0
Value of (a + b) * (c / d) is 90.0
Value of a + (b * c) / d is 50.0
Value of a + b ** d - c is 100005
```

จบบทที่ 5

บทที่ 6

เงื่อนไข การตัดสินใจ การควบคุมทิศทาง และการทำซ้ำ

(Conditions, Decisions, Control flows and Loop)



เมื่อกล่าวถึงปัญหาไม่ว่าจะเป็นปัญหาเกี่ยวกับมนุษย์หรือคอมพิวเตอร์ก็ตาม สิ่งที่หลีกเลี่ยงไม่ได้ก็คือ จะต้องพบรูปแบบของเงื่อนไข (Conditions) และต้องทำการตัดสินใจ (Decisions) อยู่เสมอๆ ยกตัวอย่างเช่น เมื่อต้องการเดินทางไปที่แห่งหนึ่ง ระหว่างการเดินทางพบทางแยกซ้าย (สมมติให้เป็นตัวแปร A) และขวา (ตัวแปร B) ผู้อ่านจะเลือกไปทางไหนดี? คำตอบนั้นจะขึ้นอยู่กับว่าเงื่อนไขทางใดดีกว่ากัน สมมติตั้งเงื่อนไขไว้ว่าพิจารณาจากระยะทาง ดังนั้นเงื่อนไขที่ใช้ตัดสินใจ คือ

เมื่อ เส้นทางซ้าย (A) น้อยกว่า เส้นทางขวา (B) และ เลือกไปทางซ้าย
ถ้าไม่ เช่นนั้น เลือกไปทางขวา

เมื่อทำการแปลงคำพูดที่กล่าวมานี้เป็น Pseudo code ได้ดังนี้

IF A < B THEN

Go to left way

ELSE

Go to right way

สำหรับการควบคุมทิศทาง (Control flows) คือ คำสั่งหรือภูมิภาคที่ใช้สำหรับควบคุมทิศทางการทำงานเพื่อให้บรรลุเป้าหมาย (ในตัวอย่างคือ สิ่งที่ไปทางซ้ายหรือขวา) แต่การจะบรรลุเป้าหมายให้ได้นั้น อาจจะทำไม่สำเร็jinครั้งเดียว จำเป็นต้องทำซ้ำหลายๆ ครั้ง (Loop)

สังเกตเห็นว่าเงื่อนไข การตัดสินใจ การควบคุมทิศทางและการทำซ้ำ เป็นสิ่งที่อยู่ใกล้ตัวเราและพบรูปแบบในชีวิตประจำวันเสมอๆ ไม่ว่าจะอยู่ที่ใดก็ตาม การเขียนโปรแกรมก็เช่นเดียวกัน พยายามที่จะจำลองปัญหาต่างๆ ในชีวิตประจำวันมาประมวลผลกับคอมพิวเตอร์ เพราะคอมพิวเตอร์สามารถทำงานที่มีความซับซ้อนได้เร็วๆ ว่ามีมนุษย์มาก

ในภาษา Python แบ่งลักษณะการควบคุมการทำงานของโปรแกรมออกเป็น 2 ประเภทหลักๆ คือ การควบคุมทิศทางแบบเลือกทำและควบคุมทิศทางแบบวนรอบหรือทำซ้ำ

1. การควบคุมทิศทางแบบเลือกทำ (Decisions, Choice, Selection)

การควบคุมทิศทางแบบเลือกทำคือ การเขียนโปรแกรมให้มีการตัดสินใจ สามารถเลือกได้ว่าจะทำ หรือไม่ตามคำสั่ง ขึ้นอยู่กับเงื่อนไขที่กำหนดขึ้นมา โดยคำสั่งสำหรับการควบคุมทิศทางแบบเลือกทำในภาษา Python มีเพียงคำสั่งเดียวคือ if โดยแบ่งออกเป็น 3 ชนิดคือ if, if...else และ nested if ดังต่อไปนี้



Note: ภาษา Python ไม่สนับสนุนการควบคุมทิศทางแบบ switch...case

การควบคุมทิศทางแบบ if

คำสั่ง if ใช้ในการสนใจมีทางเลือกให้ทำงานอยู่เพียงทางเลือกเดียว โดยถ้าตรวจสอบเงื่อนไขแล้ว เป็นจริง จึงจะทำงานตามคำสั่ง รูปแบบการเรียกใช้คำสั่ง if แสดงได้ดังนี้

```
if condition:  
    statement(s)
```

condition คือ เงื่อนไขที่กำหนดขึ้น เพื่อใช้ตัดสินว่าทำหรือไม่ตามคำสั่ง โดยเงื่อนไขจะเขียนไว้ภายใต้เครื่องหมาย () หรือไม่ก็ได้ ซึ่งเงื่อนไขอาจจะอยู่ในรูปของนิพจน์ การคำนวณ เปรียบเทียบ หรือเป็นค่าของตัวแปรก็ได้และตามด้วยเครื่องหมาย :

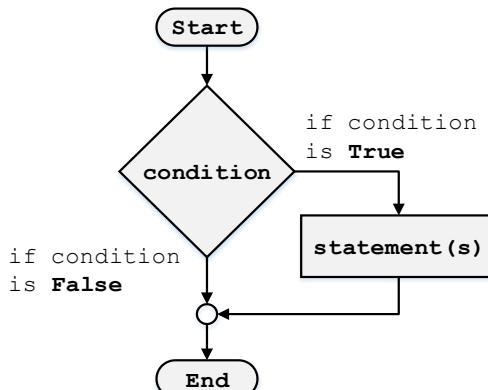
statement(s) คือ คำสั่งหรือชุดของคำสั่งที่จะให้ทำงาน เมื่อผลการตรวจสอบเงื่อนไขเป็นจริง ในภาษา Python ไม่จำเป็นต้องเขียนคำสั่ง (statement) ไว้ภายใต้ { } เมื่อเป็นภาษาซีหรือจาวา แต่ Python ใช้ การย่อหน้าเพื่อแสดงขอบเขตของคำสั่งหรือชุดคำสั่งแทน เช่น

```
if condition:  
    statement 1  
    statement 2  
    ...  
    statement n
```

คำสั่งแบบมีเงื่อนไข if เป็นคำสั่งแบบเลือกทำ โดยการเปรียบเทียบเงื่อนไขนิพจน์ทางตรรกศาสตร์ ผลลัพธ์ที่ได้จะมีค่าจริงกับเท็จเท่านั้น ถ้าเงื่อนไขเป็นจริงแล้วโปรแกรมจะเลือกทำคำสั่งที่อยู่หลังเครื่องหมาย : ทันที แต่ถ้าเป็นเท็จจะไม่มีการประมวลผลใดๆ เกิดขึ้น เมื่อเปรียบเทียบกับแผนผังจำลองการทำงานจะมีลักษณะดังภาพที่ 6.1 และตัวอย่างโปรแกรมที่ 6.1



Caution! หลังคำสั่ง if, else, for, while และ function อย่าลืมเครื่องหมาย : เป็นอันขาด



รูปที่ 6.1 แสดงแผนภาพจำลองการทำงานของคำสั่ง if



Note: ในภาษา Python ถือว่า ค่าที่ไม่ใช่ศูนย์ (0) และค่าที่ไม่ใช่ null เป็นค่าจริง และถ้าข้อมูลเป็นศูนย์ หรือ null จะอนุमานว่าเป็นเท็จทันที

ตัวอย่างโปรแกรมที่ 6.1

Program Example 6.1: if

```

1   # If command testing
2   var1 = 100;
3   if var1:           #This condition is True
4       print ("1 - Got a true expression value")
5       print (var1)
6   var2 = 0;
7   if var2:           #This condition is False
8       print ("2 - Got a true expression value")
9       print (var2)
10  print("Good bye!")
  
```



1 - Got a true expression value

100

OUTPUT

Good bye!

จากตัวอย่างโปรแกรมที่ 6.1 เริ่มต้นบรรทัดที่ 2 เป็นกำหนดค่าเริ่มต้นให้กับตัวแปร var1 = 100 บรรทัดที่ 3 ทำการตรวจสอบด้วยคำสั่ง if ว่า var1 เป็นจริงหรือไม่ (Python ถือความหมายว่า null และ 0 เท่านั้นที่เป็นเท็จ) ผลจากการเปรียบเทียบปรากฏว่าเป็นจริง ดังนั้นโปรแกรมจึงเลื่อนไปทำคำสั่งหลัง if ในบรรทัดที่ 4 และ 5 โดยพิมพ์ข้อความ "1 - Got a true expression value" และ 100 (ค่าที่เก็บอยู่ในตัวแปร var1) ต่อจากนั้นโปรแกรมจึงเลื่อนมาทำงานต่อไปในบรรทัดที่ 6 โดยกำหนดตัวแปร var2 มีค่าเท่ากับ 0 ในบรรทัดที่ 7 ทำการตรวจสอบอีกครั้งว่า var2 เป็นจริงหรือไม่ ผลที่ได้คือ เป็นเท็จ จึงทำให้โปรแกรมไม่เข้าไปประมวลผลคำสั่งหลัง if (ชุดของคำสั่งหลัง if มีสองคำสั่งคือ print โดยสังเกตจากการ

ย่อหน้าว่าทั้งสองคำสั่งย่อหน้าตรงกัน) แต่จะข้ามไปทำงานบรรทัดที่ 10 โดยสั่งพิมพ์ข้อความว่า "Good bye!" แทน พร้อมกับจบการทำงานของโปรแกรม

โจทย์ตัวอย่างและผังงาน

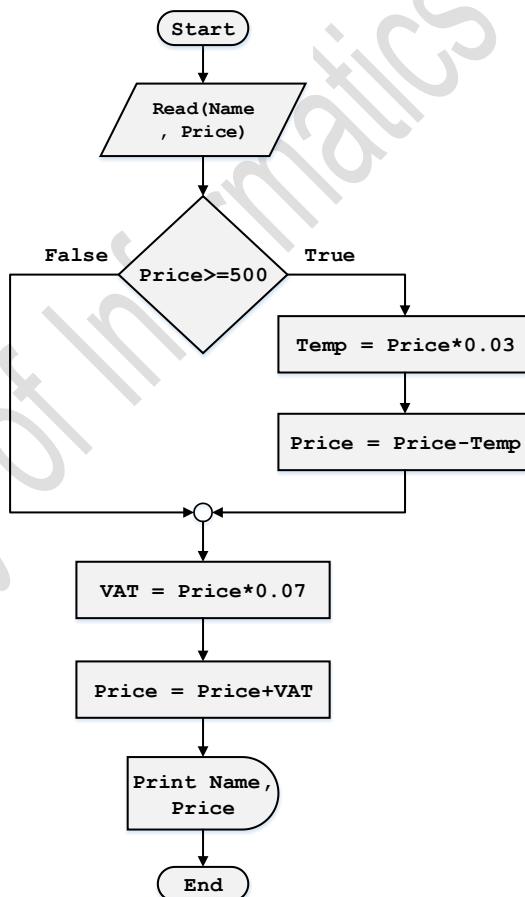
จงเขียนโปรแกรมคิดส่วนลดราคาสินค้าและภาษีมูลค่าเพิ่ม โดยรับชื่อและราคางานมาจาก
แป้นพิมพ์ ถ้าราคาสินค้าเกิน 500 บาท ทางร้านจะลดราคาให้ 3% จากนั้นจึงคำนวนหาราคาภาษีมูลค่าเพิ่ม^{ในอัตรา 7% ผลลัพธ์ที่ต้องการคือ พิมพ์ชื่อสินค้าพร้อมกับราคางานค้าที่คิดภาษีมูลค่าเพิ่มแล้ว}

ตัวอย่างอินพุต: Enter goods name : **FAN**

Enter price of **FAN** (Baht) : 350

ตัวอย่างเอาต์พุต: The price of **FAN** (inc VAT 7%) is : 374.5

ผังงานของโปรแกรมคิดส่วนลดราคาสินค้า เขียนได้ดังรูปที่ 6.2



รูปที่ 6.2 แสดงผังงานการคำนวนการลดราคาและภาษีมูลค่าเพิ่ม

ตัวอย่างโปรแกรมที่ 6.2 การคำนวนการลดราคาและภาษีมูลค่าเพิ่ม

Program Example 6.2: Discount price and VAT
--

```

1   #Calculating discount price and VAT
⇒2 name = input("Enter goods name :")
⇒3 price = float(input("Enter price of %s :" %name))
⇒4 if price >= 500:
⇒5     temp = price * 0.03
⇒6     price = price - temp
⇒7 VAT = price * 0.07
8 price = price + VAT
9 print("The price of %s (inc VAT 7%) is %.2f %s"%(name,
    price, "Baht."))

```



Enter goods name :**FAN**
 Enter price of FAN :**550.75**

OUTPUT The price of FAN (inc VAT 7%) is 571.62 Baht.

จากตัวอย่างโปรแกรมที่ 6.2 ในบรรทัดที่ 2 เริ่มต้นรับข้อมูลคือ ชื่อของสินค้าที่ต้องการคิดราคา โดยเก็บไว้ในตัวแปร name ขั้นตอนต่อไปบรรทัดที่ 3 โปรแกรมจะรอรับการป้อนข้อมูลราคาสินค้า (เป็นเลขจำนวนจริง) ราคาสินค้าจะถูกแปลงค่าเป็น float และเก็บไว้ในตัวแปร price บรรทัดที่ 4 โปรแกรมจะเปรียบเทียบค่าราคาสินค้ามากกว่า 500 หรือไม่ ถ้ามากกว่าจะประมวลคำสั่งหลัง if ในบรรทัดที่ 5 และ 6 เพื่อคำนวณส่วนลดของสินค้าให้ 3% เมื่อคำนวณส่วนลดเสร็จแล้ว โปรแกรมจะคำนวณภาษีมูลค่าเพิ่มต่อในบรรทัดที่ 7 และพิมพ์ผลลัพธ์คือ ราคาสินค้าที่ได้รับส่วนลด 3% รวมกับภาษีมูลค่าเพิ่มแล้วออกทางจอภาพในบรรทัดที่ 9

การควบคุมทิศทางแบบ if...else

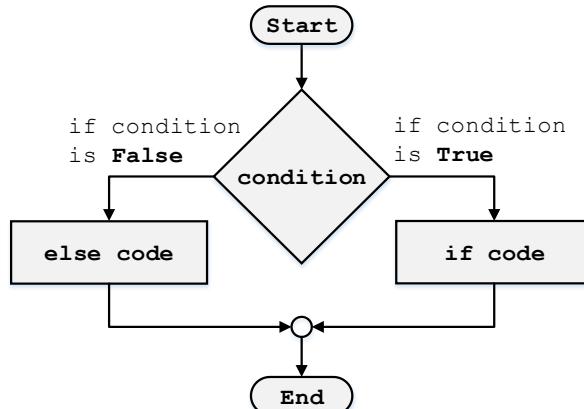
คำสั่ง if-else จะใช้ในการที่มีทางเลือกให้ทำงาน 2 ทางเลือกขึ้นไป โดยการทำงานของคำสั่ง if-else จะเริ่มจากการตรวจสอบเงื่อนไข ถ้าผลลอกมาเป็นจริง จะทำงานตามคำสั่งที่อยู่หลัง if แต่ถ้าผลของการตรวจสอบเงื่อนไขเป็นเท็จ ให้ทำงานตามคำสั่งที่อยู่หลัง else แทน รูปแบบคำสั่ง if-else แสดงดังนี้

```

if condition:
    statement(s)
else:
    statement(s)

```

แผนผังจำลองการทำงานจะมีลักษณะดังภาพที่ 6.3 และตัวอย่างโปรแกรมที่ 6.3



รูปที่ 6.3 แสดงแผนภาพจำลองการทำงานของคำสั่ง if...else

ตัวอย่างโปรแกรมที่ 6.3

Program Example 6.3: if...else

```

⇒1 var1 = 100
⇒2 if var1: #This condition is True
⇒3     print ("1 - Got a true expression value")
⇒4     print (var1)
5 else:
6     print ("1 - Got a false expression value")
7     print (var1)
⇒8 var2 = 0
⇒9 if var2: #This condition is False
10    print ("2 - Got a true expression value")
11    print (var2)
⇒12 else:
13    print ("2 - Got a false expression value")
14    print (var2)
⇒15 print ("Good bye!")

```



OUTPUT

```

1 - Got a true expression value
100
2 - Got a false expression value
0
Good bye!

```

จากตัวอย่างโปรแกรมที่ 6.3 บรรทัดที่ 1 กำหนดค่าให้ตัวแปร var1 เท่ากับ 100 เมื่อทดสอบเงื่อนไขด้วยคำสั่ง if ในบรรทัดที่ 2 ให้ผลลัพธ์เป็นจริง เพราะ var1 ไม่เท่ากับ 0 หรือ null ดังนั้นโปรแกรมจะพิมพ์ข้อความว่า "1 - Got a true expression value" (บรรทัดที่ 3) และพิมพ์ค่าของ Var1 เท่ากับ 100 (บรรทัดที่ 4) ออกทางจอภาพ จากนั้นโปรแกรมจะเลื่อนมาทำคำสั่งในบรรทัดที่ 8 คือกำหนดค่าตัวแปร var2 เท่ากับ 0 และเลื่อนมาเปรียบเทียบในเงื่อนไขด้วยคำสั่ง if อีกครั้งในบรรทัดที่ 9 ผลปรากฏว่าเงื่อนไขเป็นเท็จ เพราะ var2 เป็นเท็จ (var2 = 0) ส่งผลให้โปรแกรมข้ามไปทำงานบรรทัดที่ 13 และ 14 หลังคำสั่ง else โดยพิมพ์ข้อความ "2 - Got a false expression value" และ var2 เท่ากับ 0 ออกทาง

จากภาพ ในบรรทัดสุดท้ายของโปรแกรมจะสั่งพิมพ์ Good bye เพื่อ เพราะคำสั่งดังกล่าวไม่ได้อยู่ภายในขอบเขตของคำสั่ง if...else (สังเกตได้จากการย่อหน้าของโปรแกรม)

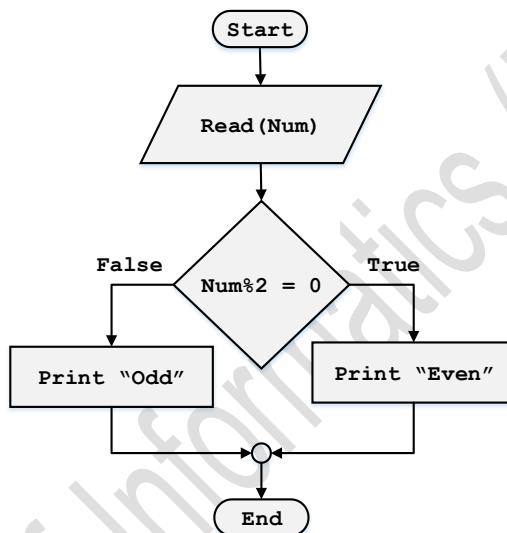
โจทย์ตัวอย่างและผังงาน

จงเขียนโปรแกรมตรวจสอบว่าตัวเลขจำนวนเต็มใดๆ เป็นเลขคู่หรือเลขคี่

ตัวอย่างอินพุต: Enter Integer number : 35

ตัวอย่างเอาต์พุต: 35 is Odd

ผังงานของโปรแกรมตรวจสอบเลขจำนวนคู่หรือคี่ ดังรูปที่ 6.4



รูปที่ 6.4 ผังงานของโปรแกรมตรวจสอบเลขจำนวนคี่คู่

ตัวอย่างโปรแกรมที่ 6.4 โปรแกรมตรวจสอบเลขจำนวนคี่คู่

Program Example 6.4: Odd or Even	
---	--

1	# Testing Odd or Even
⇒2	Num = int(input("Enter Integer Number :"))
⇒3	if Num % 2 != 0:
⇒4	print(Num, " is Odd.")
⇒5	else:
⇒6	print(Num, " is Even.")
⇒7	print("Good bye!")



OUTPUT	Enter Integer Number :35 35 is Odd. Good bye! Enter Integer Number :22 22 is Even. Good bye!
---------------	---

จากโปรแกรมที่ 6.4 บรรทัดที่ 2 โปรแกรมรับข้อมูลเป็นตัวเลขจำนวนเต็มจากแบนพิมพ์มาเก็บไว้ใน

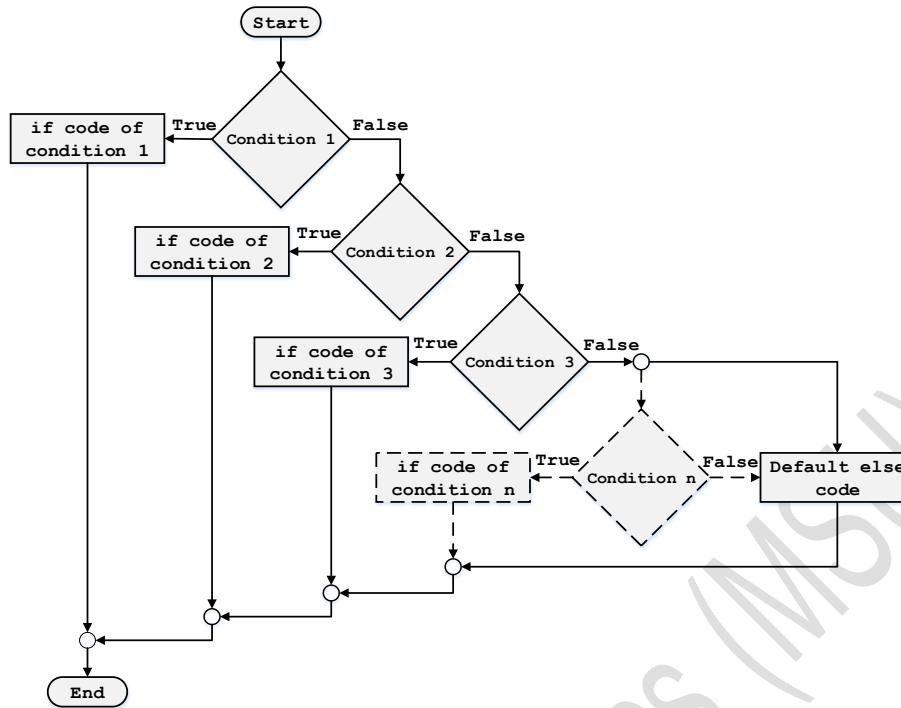
ตัวแปร Num ขั้นตอนต่อไปบรรทัดที่ 3 ทำการหารเอาเศษ (mod) ค่า Num ด้วย 2 ถ้าผลที่ได้เป็น 0 แสดงว่าเป็นเลขคู่ จะทำการสั่งหลัง else ในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า "X is Even." โดยที่ x คือ ตัวเลขจำนวนเต็มใดๆ แต่ถ้าผลที่ได้ไม่เท่ากับ 0 และแสดงว่าเป็นเลขคี่ โปรแกรมทำการสั่งหลัง if ในบรรทัดที่ 4 โดยพิมพ์ข้อความ "X is Odd." สุดท้ายโปรแกรมจะสั่งพิมพ์ "Good bye!" ในบรรทัดที่ 7 ทุกครั้ง เมื่อออกจากโปรแกรม

การควบคุมทิศทางแบบ if...elif

โครงสร้างการทำงานแบบ if...elif มีรูปแบบคำสั่งดังนี้

```
if condition 1:  
    statement(s)  
elif condition 2:  
    statement(s)  
...  
elif condition n:  
    statement(s)  
else:  
    statement(s)
```

คำสั่งรูปแบบ if..elif นี้ เป็นคำสั่งเปรียบเทียบเงื่อนไขช้อนเงื่อนไข โดยเริ่มต้นเปรียบเทียบ เงื่อนไขที่ 1 (condition 1) ถ้าเป็นเท็จ จะเลื่อนไปเปรียบเทียบกับเงื่อนไขที่ 2 (condition 2) ถ้าผลลัพธ์ การเปรียบเทียบกับเงื่อนไขที่ 2 เป็นเท็จ จึงเลื่อนไปเปรียบเทียบเงื่อนไขที่ 3 (condition 3) ต่อไปเรื่อยๆ จนกว่าจะหมดคำสั่งการเปรียบเทียบ (condition n) แต่ถ้าผลการเปรียบเทียบเงื่อนไขใดๆ แล้วผลลัพธ์ เป็นจริง จะประมวลผลคำสั่งหรือกลุ่มของคำสั่งหลังเครื่องหมาย : ของเงื่อนไขนั้นๆ เมื่อเสร็จสิ้นการ ประมวลผลคำสั่งแล้วถ้าว่าสิ้นสุดการเปรียบเทียบเงื่อนไขในกลุ่มนั้นและจบการทำงาน แผนผังจำลอง การทำงานจะมีลักษณะดังภาพที่ 6.5 และตัวอย่างโปรแกรมที่ 6.5



รูปที่ 6.5 แสดงแผนภาพจำลองการทำงานของคำสั่ง if...elif

ตัวอย่างโปรแกรมที่ 6.5

Program Example 6.5: if...elif

```

1     var = 100
2 if var == 200:
3     print ("1 - Got a true expression value")
4     print (var)
5 elif var == 150:
6     print ("2 - Got a true expression value")
7     print (var)
8 elif var == 100:
9     print ("3 - Got a true expression value")
10    print (var)
11 else:
12     print ("4 - Got a false expression value")
13     print (var)
14 print ("Good bye!")
  
```



OUTPUT

3 - Got a true expression value

100

Good bye!

จากตัวอย่างโปรแกรมที่ 6.5 บรรทัดที่ 1 กำหนดค่าให้ตัวแปร var เท่ากับ 100 ต่อจากนั้นโปรแกรมเลื่อนมาทำงานบรรทัดที่ 2 เพื่อเปรียบเทียบเงื่อนไขว่า var เท่ากับ 200 จริงหรือไม่ ผลลัพธ์ที่ได้เป็นเท็จ จึงเลื่อนมาเปรียบเทียบในบรรทัดที่ 5 ผลที่ได้มีค่าเป็นเท็จ (ตัวแปร var มีค่าไม่เท่า 150) โปรแกรมจึงเลื่อนมาเปรียบเทียบเงื่อนไขต่อในบรรทัดที่ 8 ซึ่ง var มีค่าเท่ากับ 100 เป็นจริง โปรแกรมจึงทำคำสั่ง

หลัง : ในบรรทัดที่ 9 และ 10 โดยการพิมพ์ข้อความว่า "3 - Got a true expression value" และ 100 ออกจากทางจอภาพ ต่อจากนั้นโปรแกรมจะกระโดดข้ามไปพิมพ์ข้อความว่า "Good bye!" ในบรรทัดที่ 14 และวิจัยการทำงาน

โจทย์ตัวอย่างและผังงาน

จงเขียนโปรแกรมคำนวณเกรดของนักเรียน โดยรับคะแนนจากแป้นพิมพ์ โดยมีเงื่อนไขการตัดเกรดดังนี้

ช่วงคะแนน	เกรดที่ได้
80 - 100	A
75 - 79	B+
70 - 74	B
65 - 69	C+
60 - 64	C
55 - 59	D+
50 - 54	D
0 - 49	F

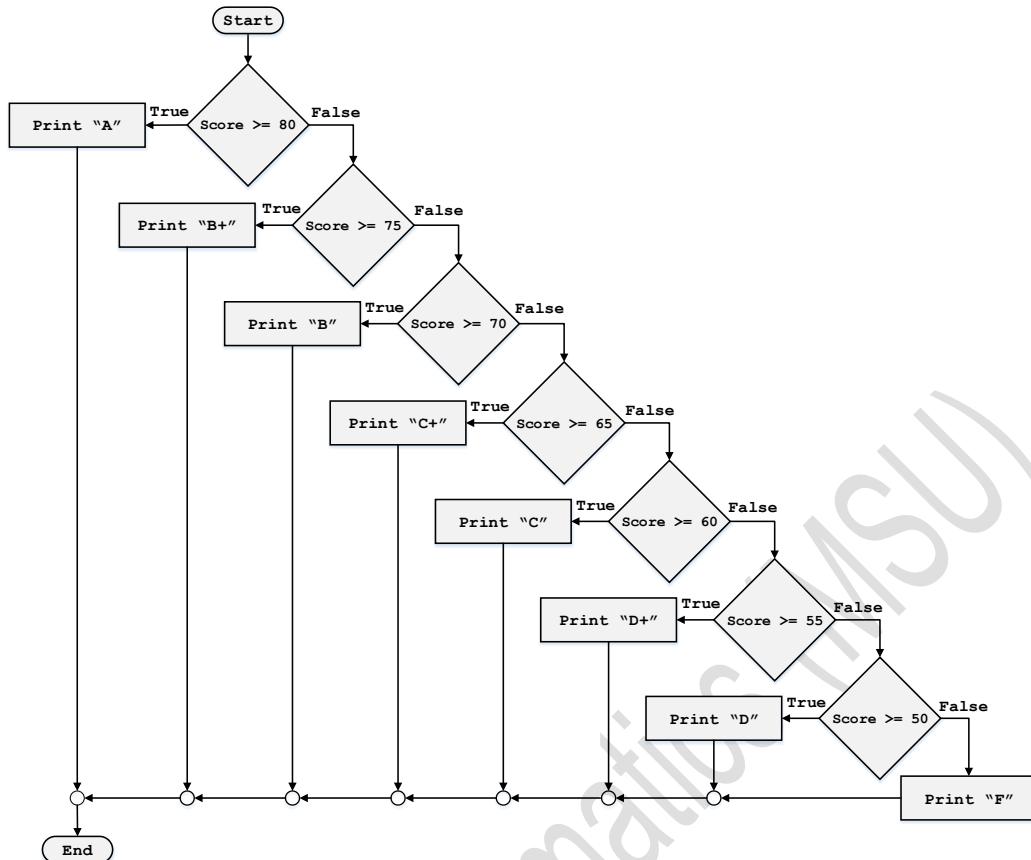
ตัวอย่างอินพุต:

Enter your score : 73

ตัวอย่างเอาต์พุต:

Your grade is B

ผังงานของโปรแกรมคำนวณเกรด ดังรูปที่ 6.6



รูปที่ 6.6 แสดง flowchart การทำงานของโปรแกรมคำนวณเกรด

ตัวอย่างโปรแกรมที่ 6.6 โปรแกรมคำนวณเกรด

Program Example 6.6: Grade evaluation

```

⇒1 Score = float(input("Enter your score :"))
⇒2 Msg = "Your grade is ";
⇒3 if Score >= 80:
4     print(Msg + "A")
⇒5 elif Score >= 75:
⇒6     print(Msg + "B+")
7 elif Score >= 70:
8     print(Msg + "B")
9 elif Score >= 65:
10    print(Msg + "C+")
11 elif Score >= 60:
12    print(Msg + "C")
13 elif Score >= 55:
14    print(Msg + "D+")
15 elif Score >= 50:
16    print(Msg + "D")
17 else:
18    print(Msg + "F")
⇒19 print("Good bye!")
  
```


OUTPUT

```
Enter your score :40
Your grade is F
Good bye!
Enter your score :75.55
Your grade is B+
Good bye!
Enter your score :82
Your grade is A
Good bye!
```

โปรแกรมตัวอย่างที่ 6.6 เริ่มต้นอ่านค่าข้อมูลจำนวนจริงจากแป้นพิมพ์ในบรรทัดที่ 1 แล้วเก็บไว้ ในตัวแปรชื่อ Score ต่อจากนั้นบรรทัดที่ 2 โปรแกรมประกาศตัวแปรชนิดสตริงชื่อ Msg มีข้อความคือ "Your grade is " สมมติว่าผู้ใช้งานป้อนข้อมูลเข้ามาคือ 75.5 โปรแกรมจะทำการเปลี่ยนเทียบกับ เงื่อนไขแรกในบรรทัดที่ 3 ผลลัพธ์ที่ได้จะเป็นเท็จ ($75.5 >= 80$) ทำให้โปรแกรมเลื่อนมาทำงานใน บรรทัดที่ 5 เพื่อเปลี่ยนเทียบค่าใน Score กับเงื่อนไขที่ 2 ($Score >= 75$) ซึ่งให้ผลลัพธ์เป็นจริง โปรแกรมจะประมวลผลคำสั่งหลัง elif ในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า "Your grade is B+" เมื่อ พิมพ์ข้อความดังกล่าวเสร็จเรียบร้อยแล้ว โปรแกรมจะกระโดดไปทำงานบรรทัดสุดท้าย (บรรทัดที่ 19) เพื่อพิมพ์ข้อความ Good bye! และจบโปรแกรม

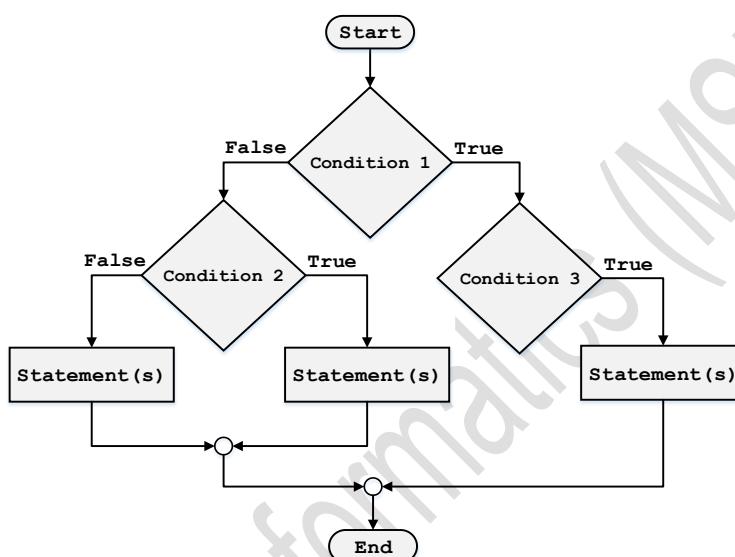
การควบคุมทิศทางแบบ nested if

โครงสร้างการทำงานแบบ nested if มีรูปแบบคำสั่งดังนี้

```
if condition (1):
    if condition (1.1):
        if condition (1.1.1):
            statement(s)
        elif (1.1.1) condition (1.1.2):
            statement(s)
        else (1.1.1):
            statement(s)
    else (1.1):
        statement(s)
else (1):
    statement(s)
```

การทำงานของ nested if นั้นจะมีลักษณะเมื่อไห้เงื่อนไขเปื่อยๆ ขึ้นอยู่กับความซับซ้อน ของโจทย์ปัญหา หากโครงสร้างการทำงานของ nested if ข้างบน เริ่มต้นจากการเปลี่ยนเทียบเงื่อนไข แรก condition (1): ถ้าเป็นเท็จจะทำคำสั่งหลัง else (1): แต่ถ้าเป็นจริง โปรแกรมจะทำคำสั่งหลัง

เครื่องหมาย : โดยหลังเครื่องหมาย : มีเงื่อนไขที่ 2 condition (1.1) ที่ต้องตรวจสอบอีกรัง ถ้าผลจากการเปรียบเทียบแล้วปรากฏว่าเป็นเท็จ โปรแกรมจะทำคำสั่งหลัง else (1.1): แต่ถ้าผลการเปรียบเทียบเป็นจริง จะเปรียบเทียบกับคำสั่ง if ในครั้งที่ 3 condition (1.1.1) แต่ถ้าผลการเปรียบเทียบในครั้งที่ 3 นี้ผลลัพธ์เป็นเท็จ จะทำการเปรียบเทียบอีกเป็นครั้งที่ 4 กับเงื่อนไข condition (1.1.2): ถ้าผลการเปรียบเทียบในครั้งที่ 4 นี้เป็นเท็จ จะประมวลผลคำสั่งหลัง else (1.1.1): แต่ถ้าเป็นจริงจะทำการประมวลผลคำสั่งหลัง condition (1.1.2): เป็นต้น สำหรับแผนผังจำลองการทำงานจะมีลักษณะดังภาพที่ 6.7 และตัวอย่างโปรแกรมที่ 6.7



รูปที่ 6.7 แสดงแผนภาพจำลองการทำงานของคำสั่ง nested if

ตัวอย่างโปรแกรมที่ 6.7

Program Example 6.7: nested if

```

⇒1 var = 100
⇒2 if var < 200:
⇒3     print("Expression value is less than 200")
⇒4     if var == 150:
5         print("Which is 150")
⇒6     elif var == 100:
⇒7         print("Which is 100")
8     elif var == 50:
9         print("Which is 50")
10    elif var < 50:
11        print("Expression value is less than 50")
12    else:
13        print ("Could not find true expression")
print("Good bye!")

```

**OUTPUT**

```
Expression value is less than 200  
Which is 100  
Good bye!
```

จากโปรแกรมที่ 6.7 บรรทัดที่ 1 กำหนดค่าเริ่มต้นให้ตัวแปร var เท่ากับ 100 จากนั้นบรรทัดที่ 2 ทำการเปรียบเทียบเงื่อนไข (`var < 200`) ผลปรากฏว่า เป็นจริง ทำให้โปรแกรมพิมพ์ข้อความว่า "Expression value is less than 200" (บรรทัดที่ 3) ออกทางจอภาพ ลำดับถัดไปในบรรทัดที่ 4 โปรแกรมทำการเปรียบเทียบเงื่อนไข `if var == 150` ผลลัพธ์คือ เป็นเท็จ โปรแกรมจึงกระโดดข้ามบรรทัดที่ 5 ไปทำการเปรียบเทียบเงื่อนไข `elif` ต่อในบรรทัดที่ 6 ผลจากการเปรียบเทียบมีค่าเป็นจริง (`var == 100`) ดังนั้นในบรรทัดที่ 7 โปรแกรมจึงสั่งพิมพ์ข้อความ "Which is 100" หลังจากพิมพ์ข้อความเสร็จ โปรแกรมจะกระโดดไปทำการคำสั่งบรรทัดที่ 14 เพื่อพิมพ์ข้อความ "Good bye!" แล้วจบการทำงาน

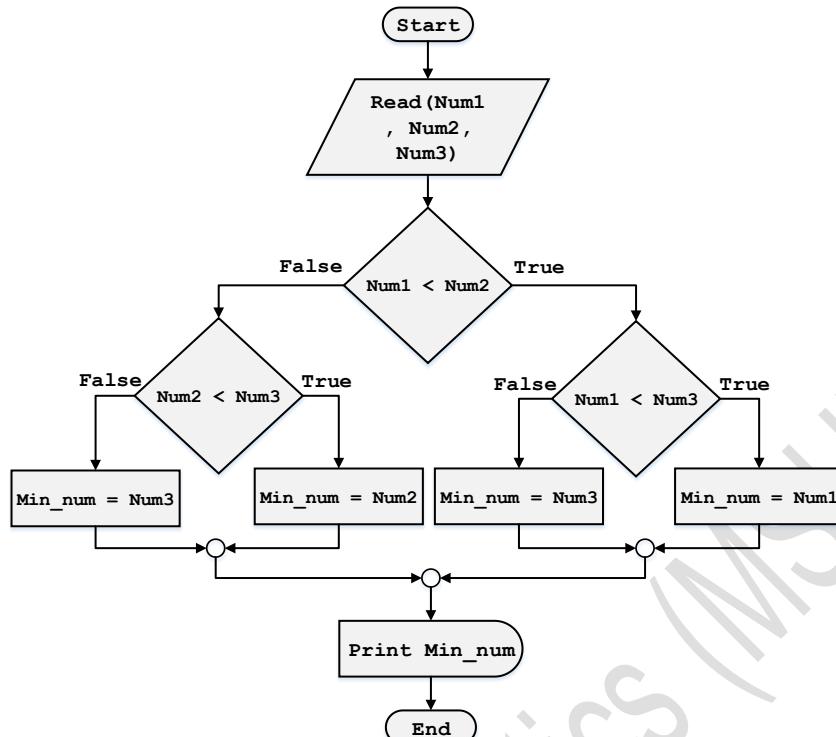
โจทย์ตัวอย่างและผังงาน

จงเขียนโปรแกรมตรวจสอบเลขจำนวนเต็มหรือจริงที่น้อยที่สุด 3 ค่า โดยรับข้อมูลจากแป้นพิมพ์

ตัวอย่างอินพุต: Enter the 1'st number : **73**
 Enter the 2'nd number : **35.7**
 Enter the 3'rd number : **53.35**

ตัวอย่างเอาต์พุต: Minimum number is **35.7**

สำหรับแผนผังจำลองการทำงานจะมีลักษณะดังภาพที่ 6.8 และตัวอย่างโปรแกรมที่ 6.8



รูปที่ 6.8 แสดงแผนภาพจำลองโปรแกรมตรวจสอบตัวเลขที่น้อยที่สุด 3 ค่า

ตัวอย่างโปรแกรมที่ 6.8 โปรแกรมตรวจสอบตัวเลขที่น้อยที่สุด 3 ค่า

Program Example 6.8: comparing 3 minimum numbers

```

1 # Comparing the minimum of 3 numbers
⇒2 num1 = float(input("Enter 1'st number :"));
3 num2 = float(input("Enter 2'nd number :"));
4 num3 = float(input("Enter 3'rd number :"));
⇒5 if num1 < num2:
6     if num1 < num3:
7         min_num = num1
8     else:
9         min_num = num3
⇒10 elif num2 < num3:
⇒11     min_num = num2
12 else:
13     min_num = num3
⇒14 print("Minimum number is ",min_num)

```



OUTPUT
Enter 1'st number :73
Enter 2'nd number :35.7
Enter 3'rd number :53.35
Minimum number is 35.7

จากโปรแกรมตัวอย่างที่ 6.8 เริ่มต้นบรรทัดที่ 2, 3 และ 4 ผู้ใช้งานป้อนข้อมูลจากแป้นพิมพ์ 3 ค่า (สมมติว่าค่าที่ป้อนเป็น 73, 35.7 และ 53.35 ตามลำดับ) เก็บไว้ในตัวแปรชื่อ num1, num2 และ num3 ขั้นตอนต่อไปบรรทัดที่ 5 โปรแกรมทำการเปรียบเทียบเงื่อนไขด้วย if ว่า num1 < num2 หรือไม่ ผลลัพธ์ที่ได้เป็นเท็จ ($73 < 35.7$) ส่งผลให้โปรแกรมเลื่อนไปประมวลผลคำสั่งในบรรทัดที่ 10 ซึ่งเป็นการ

เปรียบเทียบด้วยคำสั่ง elif num2 < num3 ผลลัพธ์ที่ได้เป็นจริง เพราะ $(35.7 < 53.35)$ โปรแกรมจะประมวลผลคำสั่งหลัง elif ในบรรทัดที่ 11 โดยกำหนดค่า min_num (ซึ่งเป็นตัวแปรที่เก็บข้อมูลตัวที่น้อยที่สุดไว้) ด้วยค่าในตัวแปร num2 ต่อจากนั้นโปรแกรมจะประมวลผลคำสั่งในบรรทัดที่ 14 โดยพิมพ์ข้อความว่า "Minimum number is" พร้อมกับค่าข้อมูลในตัวแปร min_num มาแสดงผล และจบโปรแกรม

2. การควบคุมทิศทางแบบวนรอบ หรือทำซ้ำ (Loop, Iteration)

การแก้ปัญหาต่างๆ ในชีวิตประจำวัน มักจะพบเจอกับปัญหาที่ต้องใช้ความพยายามในการแก้ปัญหาดังกล่าววนซ้ำหลายๆ ครั้ง เพื่อที่จะบรรลุเป้าหมาย เช่น ถ้าต้องการสอบให้ได้คะแนนดี จำเป็นต้องอ่านหนังสือในบทที่จะออกสอบหลายๆ รอบ ยิ่งอ่านมากยิ่งมีโอกาสที่จะได้คะแนนสอบมากตามไปด้วย หรือนักกีฬาที่ต้องการได้เหรียญทองในการแข่งขันจำเป็นต้องฝึกซ้ำแบบเดิมให้เกิดความชำนาญ ยิ่งชำนาญมากก็ยิ่งมีโอกาสประสบความสำเร็จมากตามไปด้วย เช่นเดียวกับการแก้ปัญหาทางคอมพิวเตอร์ บางปัญหานั้นจำเป็นต้องประมวลผลซ้ำไปซ้ำมาหลายๆ รอบ จนกว่าจะได้คำตอบ เช่น การหาผลรวมของจำนวนเต็มตั้งแต่ 1 – n, การหาค่า factorial, การหาค่า prime number และการคำนวณเลขลำดับอนุกรม เป็นต้น ปัญหาเหล่านี้จำเป็นต้องอาศัยเทคนิคการทำซ้ำทั้งสิ้น ภาษาไพธอน เตรียมคำสั่งในการทำซ้ำไว้ 2 คำสั่งคือ while และ for loop โดยใน 2 คำสั่งนี้ สามารถจำแนกเป็นวิธีการทำงานได้ 3 รูปแบบ while loop, for loop และ nested loop ซึ่งมีรายละเอียดดังต่อไปนี้

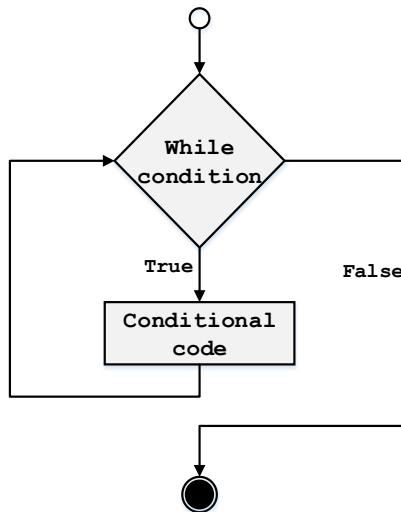
คำสั่ง While loop

While เป็นคำสั่งวนซ้ำที่มีการตรวจสอบเงื่อนไข (condition) ก่อนเข้าทำงานเสมอ เมื่อเงื่อนไขที่ทำการตรวจสอบเป็นจริง จึงจะประมวลผลคำสั่งหลัง while แต่ถ้าเงื่อนไขเป็นเท็จจะยุติการทำงานทันที สำหรับงานที่นิยมใช้ while ในการแก้ปัญหาคือ ปัญหาที่ไม่ทราบจำนวนรอบการทำงานที่แน่นอนหรือ ปัญหาที่ไม่สามารถทราบได้ร่วงหน้าว่าจะต้องใช้เวลาในการประมวลผลนานเท่าใด ส่วนใหญ่มักจะหยุดการทำงานของ while ด้วยเงื่อนไขบางประการ เช่น กดแบนพิมพ์ที่บังบอกว่าต้องการออกจากโปรแกรม เช่น ESC, q, -1, 0 เป็นต้น หรือตรวจสอบค่าในตัวแปรตัวใดตัวหนึ่งในโปรแกรมเป็นเท็จ เป็นต้น

โครงสร้างการทำงาน while loop มีรูปแบบคำสั่งดังนี้

```
while condition:  
    statement(s)
```

แผนผังจำลองการทำงานจะมีลักษณะดังภาพที่ 6.9 และตัวอย่างโปรแกรมที่ 6.9



รูปที่ 6.9 แสดงแผนภาพจำลองการทำงานของคำสั่ง while loop

ตัวอย่างโปรแกรมที่ 6.9

Program Example 6.9: while

```

1 # While loop testing
⇒2 count = 0
⇒3 while (count < 9):
⇒4     print ('The count is:', count)
⇒5     count = count + 1
⇒6 print ("Good bye!")
  
```



OUTPUT

```

The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
  
```

จากตัวอย่างโปรแกรมที่ 6.9 เริ่มต้นบรรทัดที่ 2 เป็นการกำหนดค่าให้ตัวแปร count มีค่าเท่ากับ 0 เพื่อใช้สำหรับนับค่า ขั้นตอนต่อไปบรรทัดที่ 3 โปรแกรมจะเปรียบเทียบเงื่อนไขใน while ว่า count มีค่า น้อยกว่า 9 หรือไม่ ถ้าเงื่อนไขเป็นเท็จ โปรแกรมจะข้ามไปทำงานในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า "Good bye!" ออกจอภาพ แต่ถ้าเงื่อนไขเป็นจริง ($count < 9$) โปรแกรมจะทำงานหลัง while ในบรรทัดที่ 4 โดยพิมพ์ข้อความว่า 'The count is:' พร้อมกับค่าในตัวแปร count ต่อจากนั้นในบรรทัดที่ 5 จะทำการเพิ่มค่าให้ตัวแปร count อีก 1 ต่อจากนั้นโปรแกรมจะวนกลับไปตรวจสอบเงื่อนไขของ while ในบรรทัดที่ 3 ใหม่ เป็นรอบที่ 2 และทำการประมวลผลคำสั่งตามลำดับในบรรทัดที่ 3 \Rightarrow 4 \Rightarrow 5 ไปเรื่อยๆ จนกว่าเงื่อนไขที่ while จะเป็นเท็จ ($count \geq 9$) เมื่อเงื่อนไขใน while เป็นเท็จโปรแกรมจะมาทำงานในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า "Good bye!" ก่อนจบโปรแกรมเสมอ

การวนซ้ำแบบไม่รู้จบ (The Infinite Loop)

บอยครั้งที่พับปัญหาในการใช้งาน while คือ เงื่อนไขที่ตรวจสอบไม่เป็นเท็จ ซึ่งสาเหตุมาจากการนี้ แต่ส่วนใหญ่มาจากการหลงลืมทำให้ตัวแปรที่ใช้ตรวจสอบเงื่อนไขเป็นเท็จ เช่น ในกรณีตัวอย่างโปรแกรมที่ 6.9 ถ้าหากเขียนโปรแกรมลืมเพิ่มค่าให้กับตัวแปร count จะทำให้โปรแกรมเข้าสู่สถานะที่เรียกว่า Infinite loop คือโปรแกรมทำงานไปเรื่อยๆ ไม่สามารถหยุดได้ แต่ก็มีโปรแกรมบางประเภทที่ต้องการการทำงานในลักษณะ Infinite loop อยู่เหมือนกัน เช่น โปรแกรมประเภท Client-Server ที่ผู้ให้บริการ (Server) จะรอให้บริการตลอดเวลาไม่มีวันหยุดพักหรือปิดโปรแกรมเลย สำหรับคำสั่งตัวอย่างที่ใช้ในกรณีที่ต้องการให้โปรแกรมทำงานในลักษณะ infinite loop ดังโปรแกรมที่ 6.10

ตัวอย่างโปรแกรมที่ 6.10

Program Example 6.10: *infinite loop*

```

1 # Infinite loop program
2 # When would you like to exit this program, push CTRL + C
3 var = 1
4 while var == 1 : # This constructs an infinite loop
5     num = int(input("Enter a number :"))
6     print("You entered: ", num)
7     print("Good bye!")

```



OUTPUT

```

Enter a number :394
You entered: 394
Enter a number :3994
You entered: 3994
Enter a number :Here user enter CTRL + C
Traceback (most recent call last):
  File "C:\Python34\exam6_10.py", line 5, in <module>
    num = int(input("Enter a number :"))
  File "C:\Python34\lib\idlelib\PyShell.py", line 1381, in
readline
    line = self._line_buffer or self.shell.readline()
KeyboardInterrupt

```

จากตัวอย่างโปรแกรม 6.10 เริ่มต้นบรรทัดที่ 3 กำหนดค่าให้ตัวแปร var = 1 เพื่อใช้สำหรับเปรียบเทียบเงื่อนไขก่อนเข้าทำงานใน while loop (การกำหนดเงื่อนไขเพื่อใช้เปรียบเทียบก่อนเข้าทำงานใน while loop เป็นขั้นตอนที่สำคัญมากและต้องทำเสมอ) บรรทัดที่ 4 ทำการตรวจสอบเงื่อนไขก่อนเข้าทำงานใน while ผลการตรวจสอบปรากฏว่า เป็นจริงเสมอ เพราะว่าค่าในตัวแปร var มีค่าเท่ากับ 1 โปรแกรมจึงเลื่อนไปทำคำสั่งในบรรทัดที่ 5 หลังคำสั่ง while คือคำสั่งอ่านข้อมูลจำนวนเต็มมาจากแป้นพิมพ์เก็บไว้ในตัวแปร num จากนั้นบรรทัดที่ 6 จะพิมพ์ค่าที่อยู่ในตัวแปร num ออกมายังจอภาพ และโปรแกรมจะกลับไปทำงานในบรรทัดที่ 4 อีกครั้ง ทั้งนี้ เพราะเงื่อนไขใน while ยังเป็นจริงอยู่ การทำงานจะทำซ้ำคำสั่งบรรทัดที่ 4 \Rightarrow 5 \Rightarrow 6 ไปเรื่อยๆ จนกว่าเงื่อนไขใน while จะเป็นเท็จ แต่สำหรับในกรณีนี้จะเป็นจริงตลอดไป แบบไม่มีวันจบ (Infinite loop) และคำสั่งในบรรทัดที่ 7 จะไม่ถูก

ประมวลผลโดย ถ้าผู้เขียนโปรแกรมต้องการออกจากโปรแกรมนี้ ให้กดปุ่ม CTRL + C เท่านั้น เพื่อเป็นการ Terminate โปรแกรมและโปรแกรมจะแสดงข้อความผิดพลาดอุกมาดังตัวอย่างข้างบน

การใช้คำสั่ง else ร่วมกับ while และ for

ไฟชอนจะอนุญาตให้ผู้เขียนโปรแกรมสามารถใช้ else ร่วมกับคำสั่ง while และ for ได้ (ซึ่งแนวคิดดังกล่าวไม่มีในภาษา C/C++ หรือ Java) โดยถ้าใช้คำสั่ง else กับ for แล้ว คำสั่ง else จะทำงานเมื่อการประมวลผลคำสั่งใน for loop เสร็จเรียบร้อยหมดแล้ว และเมื่อใช้งาน else กับ while โดย else จะทำงานก็ต่อเมื่อเงื่อนไขใน while loop เป็นเท็จ ตัวอย่างการใช้ else ร่วมกับ while ดังโปรแกรมที่ 6.11

ตัวอย่างโปรแกรมที่ 6.11

Program Example 6.11: while with else

```

1 # Testing else statement with while loop
2 count = 0
3 while count < 5:
4     print(count, " is less than 5 (While Loop)")
5     count = count + 1
6 else:
7     print(count, " is not less than 5(Else after exit while
loop)")
8 print("Good bye!")

```



OUTPUT

```

0 is less than 5 (While Loop)
1 is less than 5 (While Loop)
2 is less than 5 (While Loop)
3 is less than 5 (While Loop)
4 is less than 5 (While Loop)
5 is not less than 5 (Else after exit while loop)
Good bye!

```

จากตัวอย่างโปรแกรมที่ 6.11 บรรทัดที่ 2 กำหนดค่าเริ่มต้นให้ตัวแปร count เท่ากับ 0 เพื่อใช้สำหรับทำการเปรียบเทียบก่อนเข้าทำงานใน while loop ผลลัพธ์จากการเปรียบเทียบ (บรรทัดที่ 3) มีค่าเป็นจริง เพราะ count < 5 โปรแกรมจะเข้าไปประมวลผลในบรรทัดที่ 4 โดยพิมพ์ข้อความว่า "X is less than 5 (While Loop)" โดย X คือค่าที่อยู่ในตัวแปร count ในบรรทัดที่ 5 โปรแกรมทำการเพิ่มค่า conunt อีก 1 จากนั้นโปรแกรมจะวนกลับมาตรวจสอบเงื่อนไขใน while อีก (เพราะเงื่อนไขยังไม่เป็นเท็จ) ซึ่งโปรแกรมจะทำคำสั่งซ้ำในบรรทัดที่ 3 \Rightarrow 4 \Rightarrow 5 เช่นนี้ไปเรื่อยๆ จนกว่า count $>= 5$ จึงทำให้โปรแกรมยุติการทำงานใน while loop ลง และมาประมวลผลคำสั่งในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า "5 is not less than 5 (Else after exit while loop)" และตามด้วยข้อความ "Good bye!" ในบรรทัดที่ 8 ดังแสดงใน OUTPUT ของโปรแกรมด้านบน

สำหรับในกรณีที่ต้องการประมวลผลคำสั่งที่มีเพียงแค่คำสั่งเดียวเท่านั้น ต่อจากคำสั่ง while (จะมีลักษณะการทำงานคล้ายกับ if ที่ปราศจาก else) คือ ให้วาง 1 คำสั่งที่ต้องการประมวลผลอยู่ในบรรทัดเดียวกันกับ **while** ดังตัวอย่างต่อไปนี้

```
flag = True  
while flag != False: flag = False      # Single statement only  
print("Good bye!")
```

เมื่อรันโปรแกรมผลลัพธ์ที่ได้คือ

Good bye!

โจทย์ตัวอย่างและผังงาน

จงเขียนโปรแกรมคำนวณหาค่าเฉลี่ย n จำนวน โดยโปรแกรมจะนับค่าข้อมูลจากแป้นพิมพ์ไปเรื่อยๆ พร้อมคำนวณหาค่าเฉลี่ยไปพร้อมๆ กันกับการรับข้อมูล จนกว่าผู้ใช้จะพิมพ์ 0 หรือ 0.0 โปรแกรมจึงหยุดทำงาน

ตัวอย่างอินพุต/เอาต์พุต: Enter a number : **10.5**

Average of number is : **10.5**

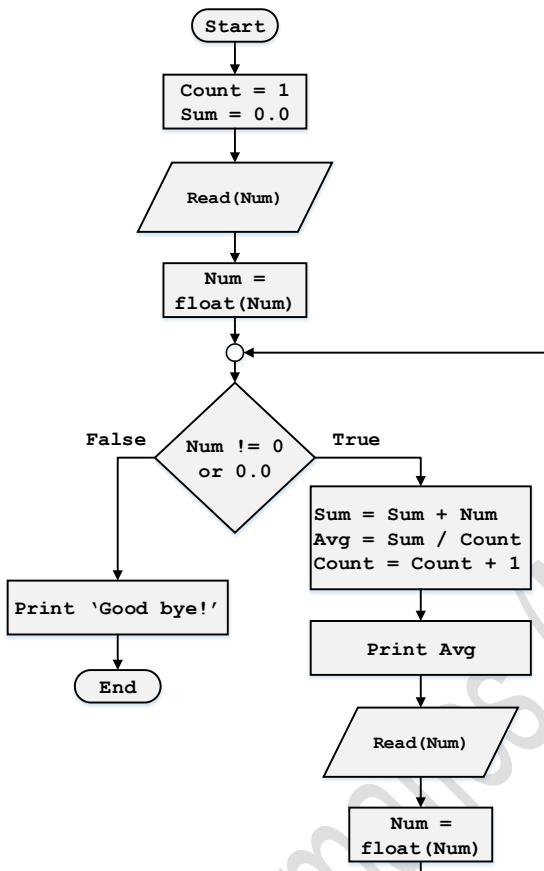
Enter a number : **5**

Average of number is : **7.75**

Enter a number : **0**

Good bye!

สำหรับแผนผังจำลองการทำงานจะมีลักษณะดังภาพที่ 6.10 และตัวอย่างโปรแกรมที่ 6.12



รูปที่ 6.10 แสดงแผนภาพจำลองโปรแกรมคำนวณหาค่าเฉลี่ย n จำนวน

ตัวอย่างโปรแกรมที่ 6.12 โปรแกรมคำนวณหาค่าเฉลี่ย n จำนวน

Program Example 6.12: The average of n numbers

```

1      # Calculting the average for N numbers
2      Count = 1
3      Sum = 0.0
4      print("To exit this program, please type 0 or 0.0 :")
5      Num = float(input("Enter a number :"))
6      while Num != 0 or Num != 0.0:
7          Sum += Num
8          Avg = Sum / Count
9          Count += 1
10         print ("Average of number is : ", Avg)
11         Num = float(input("Enter a number :"))
12         print("Good bye!")

```



OUTPUT

```

To exit this program, please type 0 or 0.0 :
Enter a number : 10.5
Average of number is : 10.5
Enter a number : 5
Average of number is : 7.75
Enter a number : 0
Good bye!

```

จากโปรแกรมที่ 6.12 บรรทัดที่ 2 และ 3 โปรแกรมกำหนดค่าเริ่มต้นให้กับตัวแปร Count เท่ากับ 1 เพื่อใช้สำหรับนับจำนวน และ Sum = 0.0 สำหรับเก็บผลรวมสะสม ในบรรทัดที่ 4 โปรแกรมพิมพ์ข้อความว่า "To exit this program, please type 0 or 0.0 :" เพื่อบอกให้ผู้ใช้ได้ทราบวิธีในการออกจากโปรแกรมดังกล่าว ขั้นตอนต่อไปบรรทัดที่ 5 โปรแกรมทำการอ่านค่าข้อมูลจากแป้นพิมพ์ ข้อมูลที่อ่านได้จะถูกแปลงให้เป็นข้อมูลชนิด float เนื่องจากการหาค่าเฉลี่ยข้อมูลจะเป็นจำนวนจริงเสมอ หลังจากนั้นในบรรทัดที่ 6 โปรแกรมทำการเปรียบเทียบเงื่อนไขในคำสั่ง while ถ้าผู้ใช้ป้อนข้อมูลที่มีค่าเป็น 0 หรือ 0.0 จะส่งผลให้การทำงานใน while loop หยุดลง และกระโดดไปทำงานบรรทัดที่ 12 โดยพิมพ์ข้อความว่า "Good bye!" พร้อมกับจบโปรแกรมทันที แต่ถ้าผู้ใช้งานป้อนค่าอื่นๆ ที่ไม่ใช่ 0 หรือ 0.0 โปรแกรมจะทำการประมวลผลหลังคำสั่ง while ในบรรทัดที่ 7 โดยเริ่มต้นคำนวณคือ นำค่า Sum + Num และเก็บไว้ในตัวแปร Sum บรรทัดที่ 8 หาค่าเฉลี่ยแล้วเก็บไว้ในตัวแปร Avg บรรทัดที่ 9 ทำการเพิ่มค่าให้ตัวแปร Count อีก 1 เสร็จแล้วพิมพ์ค่าเฉลี่ยของทางจากภาพในบรรทัดที่ 10 หลังจากพิมพ์ผลลัพธ์เสร็จแล้ว ในบรรทัดที่ 11 โปรแกรมจะรอรับข้อมูลจากแป้นพิมพ์ใหม่อีกรอบ เมื่อผู้ใช้ป้อนข้อมูลแล้ว โปรแกรมจะแปลงข้อมูลให้เป็นชนิด float เช่นเดิม เสร็จแล้วจึงกระโดดไปทำงานในบรรทัดที่ 6 เพื่อเปรียบเทียบเงื่อนไขใน while อีกรอบ โปรแกรมทำซ้ำในบรรทัดที่ 6 – 11 ในลักษณะเช่นนี้ไปเรื่อยๆ จนกว่าผู้ใช้จะป้อน 0 หรือ 0.0 โปรแกรมจึงจะยุติการทำงานใน while loop และพิมพ์ข้อความ "Good bye!" พร้อมกับจบการทำงาน

ตัวอย่างโปรแกรมที่ 6.13 จงเขียนโปรแกรมคำนวนหาค่าเฉลี่ยของการสุ่มโยนลูกเต๋าแต่ละลูกจำนวน 100 ครั้ง ทั้งหมด 2 ลูก และแสดงค่าเฉลี่ยของลูกเต๋าแต่ละลูกพร้อมกับเปรียบเทียบว่าลูกไหนมีค่าเฉลี่ยมากกว่ากัน



ตัวอย่างโปรแกรมที่ 6.13

Program Example 6.13: The average of 2 dices	
---	--

```

1      # Calculting the average of two dices
⇒2      import random
⇒3      count, avg1, avg2 = 1, 0, 0
⇒4      while count <= 100:
⇒5          dice1, dice2 = random.randrange(6)+1,
                    random.randrange(6)+1
6          avg1, avg2, count = avg1 + dice1, avg2 + dice2, count + 1
7
⇒8      print("Average of dice1 = %.2f and dice2 = %.2f"%(avg1,
                avg2))
⇒9      if avg1 > avg2:

```

```

⇒10     print("Average of dice1 > dice2")
11 else:
⇒12     print("Average of dice2 > dice1")

```



OUTPUT

```

Average of dice1 = 357.00 and dice2 = 352.00
Average of dice1 > dice2
Average of dice1 = 341.00 and dice2 = 359.00
Average of dice2 > dice1
Average of dice1 = 324.00 and dice2 = 357.00
Average of dice2 > dice1

```

จากโปรแกรมที่ 6.13 บรรทัดที่ 2 โปรแกรมนำเข้าไลบรารีชื่อ random เพื่อใช้สุ่มค่าข้อมูล

สำหรับการโยนลูกเต๋า บรรทัดที่ 3 โปรแกรมทำการกำหนดค่าเริ่มต้นให้กับตัวแปร count เท่ากับ 1 เพื่อ นับจำนวนครั้งในการโยนลูกเต๋า (โดยในโปรแกรมจะทำการโยนลูกเต๋าทั้งหมดจำนวน 100 ครั้ง) พร้อม กับกำหนดค่าให้กับตัวแปร avg1 และ avg2 มีค่าเท่ากับ 0 เพื่อเก็บค่าเฉลี่ยในการโยนลูกเต๋าแต่ละครั้ง ซึ่งจะโยนพร้อมกัน 2 ลูก เมื่อกำหนดค่าตัวแปรแล้ว บรรทัดที่ 4 โปรแกรมจะทำการตรวจสอบเงื่อนไข ของ while ว่า count <= 100 ใช่หรือไม่ ผลจากการตรวจสอบในครั้งแรก เป็นจริง เพราะ count < 100 โปรแกรมจึงประมวลผลคำสั่งหลัง while ในบรรทัดที่ 5 ด้วยการสุ่มค่าด้วยคำสั่ง randrange (จำลองการ โยนลูกเต๋า) จาก 1 – 6 ด้วยคำสั่ง random.randrange(6) + 1 (โดยปกติฟังชัน randrange จะสุ่มค่าจาก 0 ถึง n-1 ดังนั้นจำเป็นต้องบวกอีก 1 เพื่อทำให้เหมือนหน้าของลูกเต๋า) คำสั่งแรกจะเก็บค่าที่สุ่มได้ไว้ใน ตัวแปรชื่อ dice1 และตัวแปรชื่อ dice2 จะเก็บค่าการสุ่มครั้งที่สอง (การกำหนดค่าตัวแปรในบรรทัดที่ 5 ตัวแปรด้านซ้าย และข้อมูลทางด้านขวาจะต้องมีจำนวนเท่ากัน มิเช่นนั้นจะเกิดข้อผิดพลาด) จากนั้น บรรทัดที่ 6 โปรแกรมจะคำนวณหาค่าเฉลี่ย ของลูกเต๋าทั้งสองลูกแยกกัน โดย avg1 สำหรับลูกเต๋าลูกที่ 1 และ avg2 สำหรับลูกที่ 2 พร้อมกับบวกค่า count เพิ่มอีก 1

ต่อจากนั้นโปรแกรมจะกลับไปเปรียบเทียบเงื่อนไขใน while เช่นเดิม จนกว่า count จะมากกว่า 100 โปรแกรมจึงจะยุติการทำงานใน while loop และพิมพ์ค่าเฉลี่ยของลูกเต๋าในบรรทัดที่ 8 ต่อจากนั้น บรรทัดที่ 9 โปรแกรมทำการเปรียบเทียบค่าระหว่าง avg1 และ avg2 ว่าตัวแปรใดมีค่ามากกว่ากัน ถ้า avg1 มากกว่าโปรแกรมจะทำงานในบรรทัดที่ 10 โดยพิมพ์ข้อความว่า "Average of dice1 > dice2" แต่ ถ้า avg2 มากกว่า avg1 จะทำงานในบรรทัดที่ 12 โดยพิมพ์ข้อความว่า "Average of dice2 > dice1" ดังผลลัพธ์ที่แสดงข้างบน

ตัวอย่างโปรแกรมที่ 6.14 จงเขียนโปรแกรมเข้าและถอดรหัสอย่างง่าย เรียกชื่อวิธีการนี้ว่า CTOF โดย ใช้สมการเข้าและถอดรหัสดังนี้

$$F = 32 + \frac{(212 - 32)}{100} \times C$$

เมื่อ F คือตัวอักษรที่ถูกเข้ารหัสแล้ว และ C คืออักษรที่ยังไม่ได้เข้ารหัส เมื่อทำการถอดรหัสกลับจะใช้ สมการดังนี้

$$C = (F - 32)x \frac{100}{(212 - 32)}$$

ตัวอย่างอินพุต:

Enter string message : **Hello!, how are you?**

ตัวอย่างเอาต์พุต:

Encrypted message : [161.6, 213.8, 226.4, 226.4, 231.8, 91.4, 111.2, 89.6, 219.20000000000002, 231.8, 246.20000000000002, 89.6, 206.6, 237.20000000000002, 213.8, 89.6, 249.8, 231.8, 242.6, 145.4]

Decrypted string : **Hello!, how are you?**

ตัวอย่างโปรแกรมที่ 6.14

Program Example 6.14: Encryption & Decryption

```

1 # Encrypted/Decrypted program
2 # These codes for encrypting message
3 i = 0
4 encryptedMsg = []
5 msg = input("Enter string message :")
6 while i < len(msg):
7     C = ord(msg[i])
8     F = 32 + (((212 - 32)/100) * C)
9     encryptedMsg.append(F)
10    i += 1
11 print ("Encrypted message : ",encryptedMsg)
12 #These codes for decrypting message
13 i = 0
14 decryptedMsg = ""
15 while i < len(encryptedMsg):
16     F = encryptedMsg[i]
17     C = (F - 32) * (100/(212 - 32))
18     temp = chr(int(C))
19     decryptedMsg = decryptedMsg + str(temp)
20    i += 1
21 print ("Decrypted string : ",decryptedMsg)

```



OUTPUT Enter string message: **Hello!, how are you?**
 Encrypted message: [161.6, 213.8, 226.4, 226.4, 231.8, 91.4, 111.2, 89.6, 219.20000000000002, 231.8, 246.20000000000002, 89.6, 206.6, 237.20000000000002, 213.8, 89.6, 249.8, 231.8, 242.6, 145.4]
 Decrypted string: **Hello!, how are you?**

จากโปรแกรมที่ 6.14 บรรทัดที่ 3 เริ่มต้นกำหนดค่า *i* เท่ากับ 0 สำหรับนับความยาวของข้อความ และ *encryptedMsg* (บรรทัดที่ 4) เป็นค่าลิสต์ว่างเพื่อกีบข้อมูลที่เข้ารหัสแล้ว บรรทัดที่ 5 โปรแกรมทำการรับค่าข้อความจากแป้นพิมพ์และเก็บไว้ในตัวแปร *msg* หลังจากรับข้อความแล้วในบรรทัดที่ 6 โปรแกรมเริ่มการตรวจสอบค่า *i* ใน while ว่ามีค่าน้อยกว่าค่าความยาวของสตริง *msg* หรือไม่ (ความยาวของสตริงหาได้จาก *len(msg)*) เมื่อตรวจสอบแล้วผลปรากฏว่าค่า *i* มีค่าน้อยกว่าค่า

ความยาวของตรีง msg (ถ้าความยาวของสตริงมีค่า = 0 แสดงว่าไม่ได้ป้อนข้อมูล) ถ้าเป็นจริงแสดงว่า ผู้ใช้งานป้อนข้อมูลที่แบ่งพิมพ์ บรรทัดที่ 7 โปรแกรมจะทำการแปลงรหัสตัวอักษรที่ลงทะเบียนไว้เป็นจำนวนเต็ม โดยเรียกฟังชัน ord (เพื่อสมการจะใช้เลขจำนวนในการคำนวนไม่ใช้สตริง) และบรรทัดที่ 8 นำค่าที่แปลงเป็นจำนวนเต็มมาเข้าสมการเพื่อคำนวนหาค่า F บรรทัดที่ 9 ค่า F ที่คำนวนได้จะถูกนำไปพักระบบในตัวแปร encryptedMsg ก่อน โดยเก็บเป็นตัวเลขต่อท้ายไปเรื่อยๆ ด้วยเมธอด append ให้ครบทุกตัวอักษร เนื่องจากต้องรอเข้ารหัสให้ครบทุกตัวก่อนทำงานต่อไป จากนั้นบรรทัดที่ 10 ทำการเพิ่มค่าให้กับ i เพื่อนับจำนวนในรอบลูปไป ต่อจากนั้นโปรแกรมจะระดูดกลับไปทำคำสั่ง while ในบรรทัดที่ 6 อีก (เพราเวื่อนไขยังเป็นจริงอยู่) เป็นรอบที่ 2 โปรแกรมจะทำงานเรียงลำดับบรรทัดที่ 6 \Rightarrow 7 \Rightarrow 8 \Rightarrow 9 \Rightarrow 10 ในลักษณะเช่นนี้ไปเรื่อยๆ จนกว่าตัวอักษรจะถูกเข้ารหัสหมดทุกตัว นั่นคือ ค่า i เท่ากับความยาวของสตริงนั้นเอง เมื่อทำการแปลงรหัสครบตัวอักษรแล้วโปรแกรมจะยุติการทำงานใน while loop และพิมพ์ผลลัพธ์ข้อความที่เข้ารหัสแล้วในบรรทัดที่ 11 ออกทางจอภาพ ซึ่งก็จะเสร็จสิ้นกระบวนการเข้ารหัsx อาร์กิวเมนต์

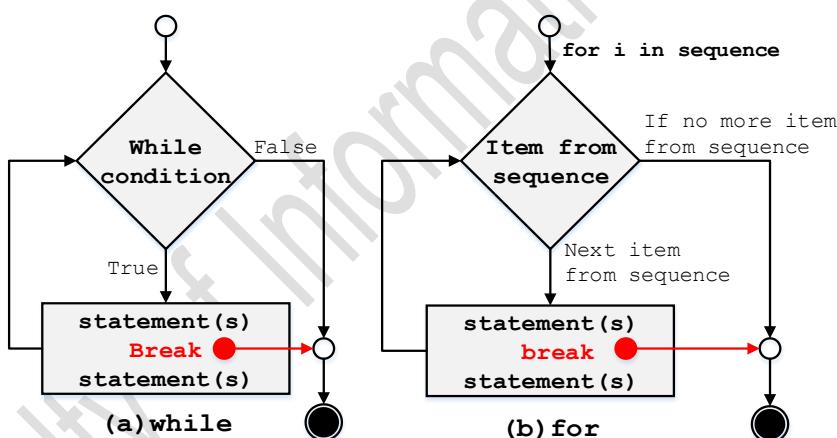
สำหรับการถอดรหัสจะเริ่มจากบรรทัดที่ 13 และ 14 โดยกำหนดค่าให้กับตัวแปร i เท่ากับ 0 อีกครั้ง (เพื่อเอาไว้นับจำนวนตัวอักษร) และตัวแปร decryptedMsg เป็นสตริงว่าง (เพื่อเก็บข้อความที่จะถูกถอดรหัสแล้ว) ลำดับถัดไปในบรรทัดที่ 15 โปรแกรมจะทำการตรวจสอบเงื่อนไขใน while ว่าข้อความว่างหรือไม่ ถ้าว่างโปรแกรมจะระดูดไปทำงานบรรทัดที่ 21 แต่ถ้าข้อความไม่ว่าง โปรแกรมจะเริ่มถอดรหัสที่ลงทะเบียนไว้กับการเข้ารหัส โดยใช้สมการการถอดรหัสในบรรทัดที่ 17 ข้อความจะถูกถอดที่ลงทะเบียนไว้และเก็บไว้ในตัวแปร temp (บรรทัดที่ 18) ในขั้นตอนนี้จำเป็นต้องทำการแปลงจากตัวเลขจำนวนจริงเป็นจำนวนเต็มก่อน ด้วยฟังชัน int(c) เพราจะสูตรที่คำนวนออกมาค่าที่ได้จะเป็นจำนวนจริง เมื่อได้จำนวนเต็มแล้วจึงทำการแปลงเป็นตัวอักษรอีกครั้ง โดยใช้ฟังชัน chr() และเก็บตัวอักษรที่ถอดรหัสแล้วในตัวแปร decryptedMsg (บรรทัดที่ 19) ในรูปแบบของสตริง พร้อมกับเพิ่มค่า i (บรรทัดที่ 20) และโปรแกรมจะระดูดไปทำงานบรรทัดที่ 15 เพื่อเปรียบเทียบเงื่อนไขใน while อีกครั้งและทำซ้ำอย่างนี้ไปเรื่อยๆ จนกว่าจะถอดรหัสข้อความครบถ้วนตัวอักษร เมื่อถอดรหัสครบถ้วนแล้ว ในบรรทัดที่ 21 โปรแกรมจะพิมพ์ข้อความที่ถอดรหัสทั้งหมดออกทางจอภาพ

คำสั่งควบคุมการทำซ้ำ (Loop control statements)

คำสั่งควบคุมการทำซ้ำถูกสร้างขึ้นเพื่อต้องการเปลี่ยนทิศทางของการประมวลผลแบบวนซ้ำ ซึ่งตามปกติแล้ว การประมวลผลจะเป็นแบบลำดับจากซ้ายไปขวาและจากด้านบนลงล่าง (Sequence) แต่สถานะการณ์บางอย่างการทำงานแบบลำดับอาจจะไม่เหมาะสม เช่น โปรแกรมต้องการค้นหาข้อความในสตริงที่มีความยาวมากๆ เมื่อค้นหาไปเรื่อยๆ ปรากฏว่าโปรแกรมพบข้อความที่ต้องการตั้งกล่าวอยู่

ระหว่างกลางของข้อความ ผู้เขียนโปรแกรมควรหยุดการทำงานนั้นๆ เมื่อจากจะทำให้ประยุคเวลาในการคำนวณ จากตัวอย่างที่กล่าวมาแล้ว เมื่อผู้เขียนโปรแกรมบังคับให้โปรแกรมหลุดออกจากคำสั่งวนซ้ำ จะส่งผลให้ตัวแปรหรืออ้อมเป็นต์ต่างๆ ที่ใช้งานอยู่ภายในขอบเขตของกรอบนั้น หมดอายุการทำงาน คือจะถูกลบออกจากหน่วยความจำไปโดยอัตโนมัติ ไฟรอนสนับสนุนคำสั่งควบคุมการทำซ้ำ 3 คำสั่งคือ break, continue และ pass

คำสั่ง break เป็นคำสั่งที่สั่งให้โปรแกรมยุติการทำงาน (Loop) ซึ่งส่งผลให้คำสั่งที่เหลือทั้งหมดหลังคำสั่ง break และอยู่ภายใต้ขอบเขตของคำสั่งทำซ้ำ ไม่ถูกประมวลผลไปด้วย เมื่อออกจากขอบเขตของคำสั่งการทำซ้ำปัจจุบันแล้ว โปรแกรมจะประมวลคำสั่งอีกครั้ง ต่อไป (ไม่ใช่การจบการทำงานของโปรแกรม จะยุติการทำงานเฉพาะคำสั่งใน Loop ปัจจุบันเท่านั้น) สำหรับการทำงานของคำสั่ง break ใน nested loop เมื่อโปรแกรมยุติการทำงานใน Loop ปัจจุบันแล้ว โปรแกรมจะทำงานต่อใน Loop ที่ครอบ Loop ปัจจุบันอยู่ต่อไป คำสั่ง break จะใช้ทำงานกับ while และ for loop เท่านั้น ไม่สามารถใช้ได้กับ if, if..else, if..elif ได้เหมือนในภาษาซี สำหรับแผนผังจำลองการทำงานของ break จะมีลักษณะดังภาพที่ 6.11



รูปที่ 6.11 แสดงแผนภาพจำลองการทำงานของคำสั่ง break

(a) break ใน while loop, (b) break ใน for loop

รูปที่ 6.11 (a) แสดงการทำงานของคำสั่ง break ใน while loop และ (b) สำหรับ for loop โปรแกรม

6.15 แสดงตัวอย่างการใช้งาน break สำหรับ while และ for loop

ตัวอย่างโปรแกรมที่ 6.15

Program Example 6.15: break for while & while

```

1 # Break for while and for loop
⇒2 for letter in 'Python':      #First example for for loop
⇒3     if letter == 'h':
⇒4         break      #Break force exit for loop
⇒5     print('Current Letter :', letter)
6

```

```

⇒7 var = 10           #Second example while loop
⇒8 while var > 0:
⇒9     print('Current variable value :', var)
⇒10    var = var - 1
⇒11    if var == 5:
⇒12        break      #Break force exit while loop
⇒13 print("Good bye!")

```


OUTPUT

```

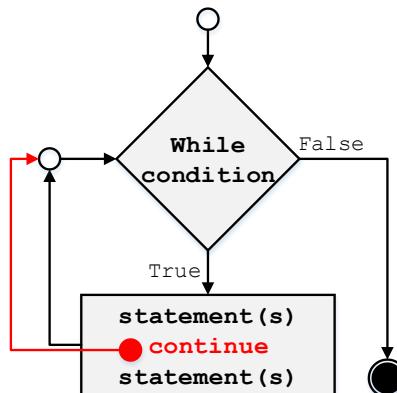
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!

```

จากตัวอย่างโปรแกรมที่ 6.15 เริ่มต้นโปรแกรมในบรรทัดที่ 2 ด้วยคำสั่งทำซ้ำแบบ for โดย for ทำการอ่านสตริง 'Python' เข้ามาครั้งละ 1 ตัวอักษร และเก็บไว้ในตัวแปร letter ต่อจากนั้นในบรรทัดที่ 3 โปรแกรมนำค่าในตัวแปร letter ไปทำการตรวจสอบเงื่อนไขกับคำสั่ง if ว่าเป็นตัวอักษร h หรือไม่ ถ้าใช่ หรือเป็นจริง โปรแกรมจะประมวลผลคำสั่ง break (บรรทัดที่ 4) เป็นผลให้โปรแกรมยุติการทำงานในลูป for ทันที และจะไปทำงานบรรทัดที่ 7 แทน แต่ถ้าเงื่อนไขที่เปรียบเทียบเป็นเท็จ คือ ตัวอักษรไม่เท่ากับ h โปรแกรมจะทำการคำสั่งในบรรทัดที่ 5 โดยพิมพ์ข้อความว่า 'Current Letter : X' เมื่อ X คือ ตัวอักษรใดๆ ออกทางจอภาพ หลังจากพิมพ์ตัวอักษรเสร็จ โปรแกรมจะกลับไปทำซ้ำในบรรทัดที่ 2 เช่นเดิม จนกว่าตัวอักษรจะเป็นตัวอักษร h หรือหมดข้อความ โปรแกรมจึงจะจบการทำงานใน for loop ผลลัพธ์ที่ได้คือ โปรแกรมพิมพ์ตัวอักษรเฉพาะ P, y และ t เท่านั้น

ต่อจากนั้นบรรทัดที่ 7 โปรแกรมกำหนดค่าตัวแปร var เท่ากับ 10 บรรทัดที่ 8 โปรแกรมทำการตรวจสอบเงื่อนไขใน while ว่า var มากกว่า 0 หรือไม่ ซึ่งในรอบแรกผลลัพธ์ที่ได้เป็นจริง โปรแกรมจะทำงานหลัง while ในบรรทัดที่ 9 พิมพ์ข้อความว่า 'Current variable value : X' โดย X คือค่าที่เก็บในตัวแปร var ออกทางจอภาพ จากนั้นบรรทัดที่ 10 โปรแกรมทำการลบค่า var ลง 1 บรรทัดที่ 11 โปรแกรมจะทำการตรวจสอบเงื่อนไขใน if ว่าค่า var เท่ากับ 5 หรือไม่ สำหรับในรอบแรก ค่าของ var จะมีค่าเท่ากับ 9 ซึ่งจะทำให้เงื่อนไข if เป็นเท็จ โปรแกรมจึงวนกลับไปทำงานในบรรทัดที่ 8 เพื่อไปตรวจสอบเงื่อนไขใน while อีกรอบ โปรแกรมจะทำงานในบรรทัดที่ 8, 9, 10 และ 11 ซ้ำในลักษณะนี้ไปเรื่อยๆ จนกว่าค่าในตัวแปร var จะมีค่าเท่ากับ 5 โปรแกรมจึงจะประมวลผลคำสั่ง break (บรรทัดที่ 12) คำสั่งดังกล่าวจะส่งผลให้โปรแกรมออกจาก while loop ทันที และพิมพ์ข้อความ "Good bye!" ก่อนจบโปรแกรม ผลลัพธ์ที่เกิดขึ้นเมื่อจบ while loop คือ โปรแกรมจะพิมพ์ข้อความเฉพาะตัวเลขจาก 9 ลงมาถึง 6 เท่านั้น เนื่องจากโปรแกรมถูกหยุดการทำงานตั้งแต่เลข 5

คำสั่ง **continue** เป็นคำสั่งที่สั่งให้โปรแกรมกลับไปเริ่มต้นใหม่ที่ต้น loop ซึ่งส่งผลให้คำสั่งที่เหลือทั้งหมดหลังคำสั่ง **continue** และอยู่ภายใต้ขอบเขตของคำสั่งทำซ้ำ จะไม่ถูกประมวลผลในรอบนั้นๆ ไปด้วย (แต่ไม่ได้ออกจากคำสั่งการทำซ้ำ) คำสั่ง **continue** จะใช้ได้ทั้ง **while** และ **for loop** สำหรับแผนผังจำลองการทำงานของ **continue** จะมีลักษณะดังภาพที่ 6.12



รูปที่ 6.12 แสดงแผนภาพจำลองการทำงานของ **continue**

โปรแกรม 6.16 แสดงตัวอย่างการใช้งาน **continue** สำหรับ **while** และ **for loop**

ตัวอย่างโปรแกรมที่ 6.16

Program Example 6.16: **continue** for while & while

```

1   # Continue for while and for loop
⇒2  for letter in 'Python':      # First example for for loop
⇒3      if letter == 'h':
⇒4          continue
⇒5      print('Current Letter :', letter)
⇒6  var = 10                      # Second example for while loop
⇒7  while var > 0:
⇒8      var = var -1
⇒9      if var == 5:
⇒10         continue
⇒11      print('Current variable value :', var)
⇒12  print("Good bye!")
  
```



OUTPUT

```

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
  
```

```
Current variable value : 1
Current variable value : 0
Good bye!
```

จากโปรแกรมตัวอย่างที่ 6.16 บรรทัดที่ 2 คำสั่ง for จะเริ่มอ่านข้อความ 'Python' เข้ามาทีละตัวอักษรแล้วเก็บไว้ในตัวแปร letter จากนั้นบรรทัดที่ 3 โปรแกรมทำการตรวจสอบค่าในตัวแปร letter ว่าเป็นอักษรตัว h หรือไม่ ถ้าใช้โปรแกรมจะทำคำสั่ง continue (ในบรรทัดที่ 4) คำสั่ง continue จะบังคับให้โปรแกรมไปเริ่มต้นทำงานที่คำสั่ง for ในรอบใหม่ โดยคำสั่งถัดจากคำสั่ง continue จะไม่ถูกประมวลผล คำสั่ง continue ส่งผลให้โปรแกรมยกเลิกไม่พิมพ์ตัวอักษร h ออกจากทางจอภาพ (บรรทัดที่ 5) สำหรับตัวอักษรอื่นๆ ที่ไม่ใช่ตัว h จะถูกพิมพ์ออกจากทางจอภาพทั้งหมด

สำหรับโปรแกรมในส่วนของ while loop จะเริ่มต้นในบรรทัดที่ 6 โดยการกำหนดค่าตัวแปร var เท่ากับ 10 ในบรรทัดที่ 7 โปรแกรมจะตรวจสอบเงื่อนไขด้วยคำสั่ง while ว่า var > 0 หรือไม่ เมื่อเงื่อนไขเป็นจริงโปรแกรมจะทำงานในบรรทัดที่ 8 แต่ถ้าเป็นเท็จ โปรแกรมจะทำคำสั่งในบรรทัดที่ 12 สำหรับในบรรทัดที่ 8 โปรแกรมจะลดค่าตัวแปร var ลง 1 จากนั้นบรรทัดที่ 9 โปรแกรมจะตรวจสอบเงื่อนไขใน if ว่า var เท่ากับ 5 หรือไม่ เมื่อผลของการเบรี่ยบเทียบเป็นจริง โปรแกรมจะประมวลผลคำสั่ง continue ในบรรทัดที่ 10 ผลจากคำสั่งดังกล่าวจะบังคับให้โปรแกรมไปเริ่มต้นทำงานใหม่ที่จุดเริ่มต้นของ while loop ทันที ผลลัพธ์ที่ได้คือ โปรแกรมจะไม่พิมพ์ตัวเลข 5 ออกจากทางจอภาพ สำหรับตัวเลขอื่นๆ จะถูกพิมพ์ออกจากทางทั้งหมด

คำสั่ง pass (ไม่มีการประมวลผลใดๆ เกิดขึ้น) เป็นคำสั่งที่ไม่เพื่อรักษาโครงสร้างหรือความหมายของโปรแกรมไว้ เช่น กรณีที่ผู้เขียนโปรแกรมกำลังเขียนโปรแกรมอยู่ แต่ปรากฏว่าในบางจุดของโปรแกรม ผู้เขียนยังไม่แน่ใจว่าจะดำเนินการต่ออย่างไรและต้องการยกเว้นโปรแกรมตรงส่วนนี้ไว้ก่อน (เช่น comment ก็พอใช้ได้เหมือนกัน) แล้วจึงค่อยมาแก้ไขในภายหลัง ผู้เขียนโปรแกรมสามารถบรรจุคำสั่ง pass นี้ไว้เพื่อให้โปรแกรมสามารถทดสอบรันโปรแกรมได้และประมวลผลโปรแกรมในส่วนที่ละเว้นไว้ด้วย ซึ่งคำสั่ง pass จะไม่มีการประมวลผลใดๆ เกิดขึ้น (คล้ายคำสั่ง No operation ในภาษาแอสแซมบลี) และเมื่อไม่ต้องการใช้งานคำสั่งดังกล่าวแล้ว ไม่จำเป็นต้องลบทิ้งก็ได้ (แต่ควรลบจะดีกว่า) สำหรับตัวอย่างการใช้งานคำสั่ง pass แสดงในตัวอย่างโปรแกรมที่ 6.17



Tips: เมื่อป้อนคำสั่ง pass และกด Enter โปรแกรมจะออกจากการตรวจสอบคำสั่งที่กำลังเขียนโปรแกรมทันที เช่น เมื่อกำลังเขียนคำสั่งอยู่ใน while loop เมื่อป้อนคำสั่ง pass เครื่องเซอร์จะมาปรากฏในตำแหน่งเดียวกับคำสั่ง while ทันที

ตัวอย่างโปรแกรมที่ 6.17

Program Example 6.17: pass

```
1 # Testing pass command
2 for letter in 'Python':
    if letter == 'h':
```

⇒3

```

⇒4     pass
⇒5     print('This is pass block')
6      print('Current Letter :', letter)
7      print("Good bye!")

```


OUTPUT

```

Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!

```

จากตัวอย่างโปรแกรมที่ 6.17 การทำงานเริ่มต้นในบรรทัดที่ 2 คำสั่ง `for` จะอ่านตัวอักษรทีละตัว จากข้อความ 'Python' โดยตัวอักษรแต่ละตัวจะถูกตรวจสอบว่าเป็นตัวอักษรเป็นตัว `h` หรือไม่ ถ้าผลจากการตรวจสอบเป็นเท็จ โปรแกรมจะพิมพ์ตัวอักษรทีละตัว เริ่มตั้งแต่ `P`, `y`, `t` ตามลำดับ แต่เมื่อเปรียบเทียบแล้วเป็นตัวอักษร `h` โปรแกรมจะประมวลผลคำสั่ง `pass` (บรรทัดที่ 4) ซึ่งเป็นคำสั่งที่ไม่มีผลกระทบใดๆ กับโปรแกรม แต่เป็นการบ่งบอกให้ผู้เขียนโปรแกรมทราบว่าโปรแกรมตรงส่วนนี้ยังไม่เสร็จสมบูรณ์ (อาจจะมีการปรับปรุงเพิ่มเติมในภายหลัง) และต้องการให้ประมวลผลโปรแกรมตรงส่วนนี้ด้วย เพราะต้องการทดสอบการทำงานของคำสั่ง `if` ว่าทำงานได้จริงหรือไม่ (การใช้ `comment` แทนที่เป็นทางเลือกที่ดี แต่ `comment` มีหน้าที่หลักในการอธิบายการทำงานของโปรแกรมมากกว่า) เพื่อให้การทำงานของคำสั่ง `pass` เห็นได้ชัดเจนขึ้น (เพราะคำสั่ง `pass` จะไม่แสดงผลใดๆ ให้เห็น) ผู้เขียนจึงเพิ่มคำสั่งที่พิมพ์ข้อความว่า 'This is pass block' ในบรรทัดที่ 5 เข้ามาช่วยเพื่อให้เห็นผลของการประมวลผลคำสั่ง `pass` ได้ชัดขึ้น ผลลัพธ์เมื่อสั่งรันโปรแกรมแล้วแสดงในตัวอย่างข้างบน



Note: คำสั่ง `switch`, `do-while`, `forecach` ไม่มีให้ใช้งานในภาษา Python

คำสั่ง `for loop`

คำสั่ง `for` เป็นคำสั่งที่ใช้สำหรับการทำซ้ำ เช่นเดียวกับ `while` และต้องมีการตรวจสอบเงื่อนไขก่อนเข้าลูปเหมือนกัน แต่แตกต่างกันตรงที่ `for` จะตรวจสอบรายการการแบบลำดับแทน (`Item of sequence`) เช่น ข้อมูลชนิดสตริง ลิสต์ หรือทัพเพิล เป็นต้น โครงสร้างการทำงาน `for loop` มีรูปแบบคำสั่งดังนี้

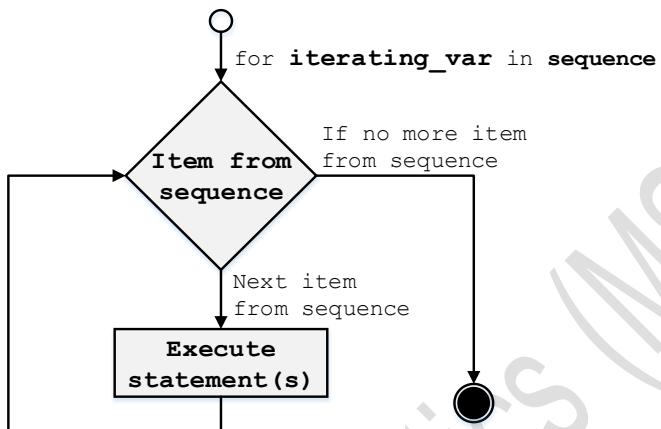
```

for iterating_var in sequence:
    statements(s)

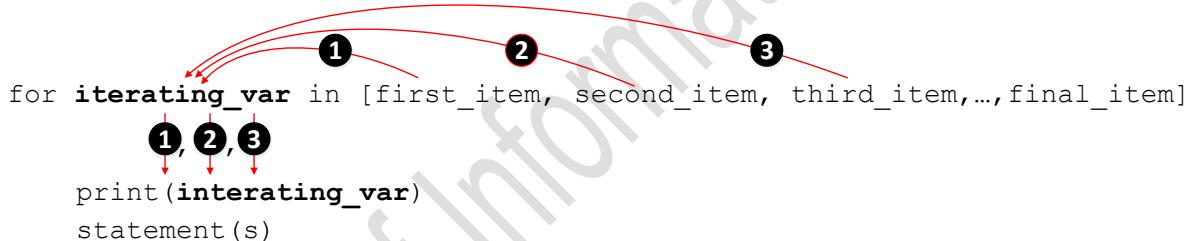
```

โดย `iterating_var` คือตัวแปรที่ใช้สำหรับรับค่าที่จะค่าเพื่อนำมาประมวลผล จากข้อมูลที่อยู่ในตัวแปร `sequence` เมื่อข้อมูลในตัวแปร `sequence` เป็นชนิดลิสต์ คำสั่ง `for` จะดึงข้อมูลในตำแหน่งแรก

ของลิสต์或มาเก็บไว้ใน `iterating_var` หลังจากนั้นจะเริ่มทำคำสั่งใน `statement(s)` เมื่อคำสั่งใน `statement(s)` หมดแล้ว การควบคุมจะกลับไปเริ่มต้นใหม่ที่ `for` และดึงข้อมูลในลิสต์ลำดับถัดไปมาทำงาน การทำงานจะเป็นไปในลักษณะเช่นนี้ไปเรื่อยๆ จนกว่าข้อมูลในลิสต์จะหมด `for` จึงจะยุติการทำงาน สำหรับแผนผังจำลองการทำงานของ `for` และตัวอย่างการดึงข้อมูลสมาชิกในรายการ ดังภาพที่ 6.13 และ 6.14 ตามลำดับ



รูปที่ 6.13 แสดงแผนภาพจำลองการทำงานของ `for`



รูปที่ 6.14 แสดงการดึงข้อมูลสมาชิกจากตัวแปรชนิดลิสต์ของ `for loop`

จากรูปที่ 6.14 แสดงวิธีการดึงข้อมูลจากตัวแปรลิสต์เข้ามาทำงานใน `for` อันดับแรก `for` จะทำการอ่าน `first_item` ซึ่งเป็นข้อมูลสมาชิกตัวแรกในลิสต์มาเก็บไว้ใน ① `iterating_var` จากนั้นโปรแกรมจะสั่งพิมพ์ค่าที่อยู่ในตัวแปร `iterating_var` ออกจอภาพ และทำคำสั่ง `statement(s)` จนหมด เมื่อทำคำสั่งใน `statement(s)` ดังกล่าวเสร็จแล้ว คำสั่ง `for` จะทำการดึงข้อมูลสมาชิกจากลิสต์ตัวถัดไป ② (`second_item`) เข้ามาทับข้อมูลตัวแรกที่อยู่ใน `iterating_var` จากนั้นโปรแกรมจะทำการพิมพ์ค่าในตัวแปร `iterating_var` และทำคำสั่ง `statement(s)` เช่นเดิม โปรแกรมจะทำการอ่านข้อมูลสมาชิกตัวถัดไปมาทำงานเรื่อยๆ จนกว่าจะไม่มีข้อมูลอยู่ในลิสต์ `for` จึงจะยุติการทำงาน สำหรับตัวอย่างการใช้ `for` แสดงดังโปรแกรมที่ 6.18

ตัวอย่างโปรแกรมที่ 6.18

Program Example 6.18: `for`

1	# Testing for loop
⇒2	for letter in 'Python': # First Example

```

⇒3 print('Current Letter :', letter)
⇒4 fruits = ['banana', 'apple', 'mango']
⇒5 for fruit in fruits:           # Second Example
⇒6     print('Current fruit :', fruit)
7 print("Good bye!")

```


OUTPUT

```

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!

```

จากโปรแกรมตัวอย่างที่ 6.18 บรรทัดที่ 2 คำสั่ง for เริ่มอ่านข้อมูลสตริงเข้ามาทีละตัวอักษร และเก็บไว้ในตัวแปร letter ต่อจากนั้นโปรแกรมจะทำงานในบรรทัดที่ 3 โดยการพิมพ์ตัวอักษรที่เก็บอยู่ ในตัวแปร letter ออกมานั่นเอง ซึ่งตัวอักษรตัวแรกในสตริงชุดนี้คือตัว 'P' หลังจากนั้นโปรแกรมจะกลับไปเริ่มต้นอ่านข้อมูลใหม่ ซึ่งอักษรตัวที่ 2 ในสตริงคือ 'y' โดยเก็บทับตัวอักษรเดิมที่อยู่ในตัวแปร letter จากนั้นโปรแกรมจะพิมพ์ตัวอักษรอ กองทางจuxtaposition โปรแกรมจะทำงานในลักษณะอย่างนี้ไปเรื่อยๆ จนถึงอักษรตัวสุดท้ายในสตริง (ในที่นี้คือ n) จึงจะยุติการทำงานของ for ผลลัพธ์ที่ได้คือ ข้อความ 'Python'

บรรทัดที่ 4 โปรแกรมจะกำหนดค่าข้อมูลให้กับตัวแปรลิสต์ชื่อ fruits คือ 'banana', 'apple' และ 'mango' ตามลำดับ ต่อจากนั้นในบรรทัดที่ 5 คำสั่ง for จะทำการอ่านข้อมูลสมาชิกตำแหน่งแรกจากตัวแปร fruits คือ 'banana' มาเก็บไว้ในตัวแปร fruit ในบรรทัดที่ 6 โปรแกรมจะพิมพ์ข้อมูลที่เก็บอยู่ในตัวแปร fruit ออกจากการผลลัพธ์คือ 'banana' เมื่อพิมพ์ข้อมูลเสร็จ โปรแกรมจะกลับไปเริ่มต้นคำสั่ง for อีกครั้ง โดยโปรแกรมจะทำคำสั่งเหมือนเดิมอย่างนี้ไปเรื่อยๆ จนกว่าข้อมูลของสมาชิกที่อยู่ในตัวแปร fruits จะหมด คำสั่ง for จึงจะยุติการทำงาน ผลลัพธ์ที่ได้คือ ข้อความ 'banana', 'apple' และ 'mango' ตามลำดับ

การอ้างถึงข้อมูลสมาชิกโดยการใช้ตำแหน่งของ for loop

คำสั่ง range เป็นคำสั่งที่ช่วยสร้างช่วงของข้อมูล เช่น เมื่อใช้คำสั่ง range(5) ช่วงข้อมูลที่ได้คือ 0, 1, 2, 3, 4 ดังนั้นเราสามารถนำ range มาประยุกต์ใช้กับ for ได้ โดยข้อมูลที่สร้างโดย range จะถูกนำไปใช้เป็นตัวชี้ตำแหน่งของการอ้างถึงข้อมูลได้ ดังตัวอย่างโปรแกรมที่ 6.19

ตัวอย่างโปรแกรมที่ 6.19

Program Example 6.19: for and range
--

1 # for loop and index

```

⇒2 fruits = ['banana', 'apple', 'mango']
⇒3 for index in range(len(fruits)):
⇒4     print('Current fruit :', fruits[index])
5    print("Good bye!")

```



OUTPUT

```

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!

```

จากโปรแกรมตัวอย่างที่ 6.19 บรรทัดที่ 2 เป็นการกำหนดค่าให้กับตัวแปรลิสต์ชื่อ fruits มีค่าเป็น 'banana', 'apple' และ 'mango' ตามลำดับ ต่อจากนั้นในบรรทัดที่ 3 คำสั่ง for จะสร้างช่วงของข้อมูลโดยอาศัยคำสั่ง range สำหรับช่วงข้อมูลที่เกิดจากคำสั่ง range คือค่า 0, 1, 2 (ซึ่งเกิดจากการหาจำนวนสมาชิกของตัวแปร fruits โดยใช้ฟังชัน len(fruits) เพราะ fruits มีสมาชิก 3 ตัว) ลำดับต่อไปคำสั่ง for จะทำการอ่านข้อมูลที่สร้างโดย range มาทีละค่า โดยเริ่มต้นที่ 0 และเก็บค่าดังกล่าวไว้ในตัวแปร index (index = 0) ต่อจากนั้นในบรรทัดที่ 4 โปรแกรมจะพิมพ์ข้อความว่า 'Current fruit : ' พร้อมกับข้อมูลที่ดึงมาจากตัวแปร fruits ในตำแหน่งที่ index มีค่าเท่ากับ 0 คือ fruits[index] = fruits[0] = 'banana' ออกแบบของการ ลำดับต่อไปโปรแกรมจะกลับไปเริ่มต้นคำสั่ง for ใหม่ เพราะข้อมูลใน range ยังไม่หมด โดยดึงค่าข้อมูลในตำแหน่งถัดไปของตัวแปร fruits มาแสดง การทำงานจะทำซ้ำอย่างนี้ไปเรื่อยๆ จนกว่าสมาชิกใน range จะหมด ผลจากการทำงานของโปรแกรมคือ โปรแกรมจะพิมพ์ข้อความ 'banana', 'apple', 'mango' สำหรับบรรทัดสุดท้ายโปรแกรมจะพิมพ์ข้อความ 'Good bye!' ก่อนจบโปรแกรมเสมอ

ฟังชัน range จากที่กล่าวไว้แล้วว่า for นั้นนิยมใช้งานกับคำสั่ง range เสมอ ดังนั้นในย่อหน้านี้จะกล่าวถึงคำสั่ง range เพิ่มเติมอีกเล็กน้อย เพื่อให้การใช้คำสั่ง range ร่วมกับ for ได้ดีขึ้น โดยปกติคำสั่ง range จะมีรูปแบบคำสั่ง 4 แบบดังนี้

① range (x), ② range (x, y), ③ range (x, y, i), ④ range (y, x, -i)

range (x) แบบที่ ① จะสร้างชุดของข้อมูลเริ่มต้นจาก 0 ถึง (x -1) โดยเพิ่มขึ้นครั้งละ 1 เช่น เมื่อเรียกใช้ฟังชัน range(6) ข้อมูลที่ถูกสร้างขึ้นคือ [0, 1, 2, 3, 4, 5]

range (x, y) แบบที่ ② จะสร้างชุดของข้อมูลเริ่มต้นจาก x ถึง (y -1) โดยเพิ่มขึ้นครั้งละ 1 เช่น เมื่อเรียกใช้ฟังชัน range(3, 10) ข้อมูลที่ถูกสร้างขึ้นคือ [3, 4, 5, 6, 7, 8, 9]

range (x, y, i) แบบที่ ③ จะสร้างชุดของข้อมูลเริ่มต้นจาก x ถึง (y -1) โดยเพิ่มขึ้นครั้งละ i เช่น เมื่อเรียกใช้ฟังชัน range(3, 15, 2) ข้อมูลที่ถูกสร้างขึ้นคือ [3, 5, 7, 9, 11, 13] มีข้อสังเกตสำหรับค่า y ตัวสุดท้าย คือ 15 จะไม่ถูกนำมาใส่ไว้ในรายการด้วย เนื่องจากติดเงื่อนไขที่ค่า y = y - 1 ดังนั้นค่า y ที่ได้คือ 14 โปรแกรมจึงตัดทั้ง 14 และ 15 ทิ้งไปด้วย เพราะไม่อยู่ในเงื่อนไขทั้งคู่

`range (y, x, -i)` แบบที่ ❸ จะสร้างชุดของข้อมูลแบบย้อนหลัง จาก y ถึง $(x - 1)$ โดยลดลงครั้งละ $-i$ เช่น เมื่อเรียกใช้ฟังชัน `range(15, 3, -2)` ข้อมูลที่ถูกสร้างขึ้นคือ `[15, 13, 11, 9, 7, 5]` ตามลำดับ ถ้าต้องการให้ลดค่าทีละ 1 โดยเริ่มตั้งแต่ 15 ถอยหลังไปจนถึง 1 มีรูปแบบคือ `range(15, 0, -1)` หรือถ้าต้องการให้เป็นค่าที่ติดลบ โดยเริ่มตั้งแต่ 0 ลงไปถึง -4 และลดค่าครั้งละ 1 มีรูปแบบคือ `range(0, -5, -1)` สำหรับตัวอย่างเพิ่มเติมการใช้ `for` กับ `range` ดังตัวอย่างโปรแกรมที่ 6.20

ตัวอย่างโปรแกรมที่ 6.20

Program Example 6.20: `for` and `range` examples

```

1   # explain function range
2   ↪2   import sys, random
3   ↪3   sys.stdout.write("Show range (6) = ")
4   for i in range(6):
5       sys.stdout.write(str(i) + " ")
6   print("\n" + "-" * 70)
7   sys.stdout.write("Show range (1, 7) = ")
8   for j in range(1, 7):
9       sys.stdout.write(str(j) + " ")
10  print("\n" + "-" * 70)
11  sys.stdout.write("Show range (1, 36, 2) = ")
12  for o in range(1, 36, 2):
13      sys.stdout.write(str(o) + " ")
14  print("\n" + "-" * 70)
15
16  sys.stdout.write("Show range [1, 3,..., 36] =")
17  for r in [1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36]:
18      sys.stdout.write(str(r) + " ")
19  print("\n" + "-" * 70)
20  sys.stdout.write("Show multiple range = ")
21  for d1 in range(2):
22      for d2 in range(2):
23          print (d1 + 1, "+", d2+1, '=', d1+d2+2)
24  print("\n" + "-" * 70)
25  sys.stdout.write("Show range + random = ")
26  for i in range(5):
27      d1= random.randrange(6)+1
28      d2= random.randrange(6)+1
29      print (d1+d2)
30  print("Show range backward")
31  ↪31  for i in range(5, 0, -1):
32      print(i)

```



OUTPUT

```

Show range (6) = 0 1 2 3 4 5
-----
Show range (1, 7) = 1 2 3 4 5 6
-----
Show range (1,36,2) = 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
-----
Show range [1, 3,..., 36] =1 3 5 7 9 12 14 16 18 19 21 23 25 27 30 32
34 36
-----
```

```

Show multiple range = 1 + 1 = 2
1 + 2 = 3
2 + 1 = 3
2 + 2 = 4
-----
Show range + random = 8
2
7
8
8
Show range backward
5
4
3
2
1

```

จากตัวอย่างโปรแกรมที่ 6.20 บรรทัดที่ 2 คือคำสั่งนำเข้าไลบรารี sys เพื่อเรียกใช้เมธอด sys.stdout.write() เนื่องจากคำสั่ง print จะไม่ยอมให้ผู้เขียนโปรแกรมพิมพ์ข้อความเว้นวรรคติดกันในแต่เดียว กันได้ (ปกติจะขึ้นบรรทัดใหม่) ดังนั้นจึงจำเป็นต้องนำเมธอด sys.stdout.write() มาช่วยในการแสดงผลข้อมูลให้อยู่ในบรรทัดเดียว กัน และ random สำหรับสุ่มชุดข้อมูลตัวเลข บรรทัดที่ 6, 10, 14, 19 และ 24 คือคำสั่ง print("\n" + "-" * 70) เป็นการสั่งให้ขึ้นบรรทัดใหม่ โดยใช้รหัส '\n' และทำการพิมพ์ '-' จำนวน 70 ครั้งติดต่อกันในบรรทัดเดียว สำหรับคำสั่งในบรรทัดที่ 27 คือ random.randrange(6) + 1 เป็นการสุ่มค่าได้ค่าหนึ่งตั้งแต่ 1 – 6 (ปกติจะสุ่มค่า 0 – 5 แต่ในที่นี่บวก 1 เพิ่มเข้าไป ดังนั้นค่าที่ได้จะเป็น 1 – 6) ในบางกรณี เมื่อผู้เขียนโปรแกรมต้องการชุดของข้อมูลแบบย้อนกลับ จึงสามารถหาด้วยความสามารถทำได้โดย กำหนดค่าของตัวแปร i ให้เป็นค่าลบและสลับค่าของตัวแปร x กับ y ดังตัวอย่าง โปรแกรมในบรรทัดที่ 31

การใช้ else statement กับ for loop

“เพื่อนอนนุญาตให้ใช้ else กับ for loop” ได้ ซึ่งแตกต่างจากภาษาอื่นๆ โดยทั่วไป เช่นภาษาซี หรือจาวา เป้าหมายสำคัญของการใช้ else กับ for นั้นเพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรมในการนับที่เงื่อนไขใน for เป็นเท็จ โดยปกติจะออกจากคำสั่ง for ไปโดยอัตโนมัติ บางครั้งผู้เขียนโปรแกรมไม่ทราบเลยว่าเกิดข้อผิดพลาดอะไรขึ้นในลูป for ดังนั้นคำสั่ง else จึงช่วยให้ผู้เขียนโปรแกรมสามารถตรวจสอบความผิดปกติใน for ได้ถูกทางหนึ่งหรือผู้เขียนโปรแกรมต้องการทำงานบางอย่างหลังจากการทำงานใน for เรียบร้อยแล้ว สำหรับการใช้ else กับ for และ while นั้นมีข้อพิจารณาดังนี้

- 1) คำสั่ง else เมื่อถูกใช้กับ for loop: คำสั่ง else จะถูกประมวลผลเมื่อ คำสั่งใน for ถูกประมวลผลครบหมดแล้ว
- 2) คำสั่ง else เมื่อถูกใช้กับ while loop: คำสั่ง else จะถูกประมวลผลเมื่อ เงื่อนไขใน while เป็นเท็จ

พิจารณาตัวอย่างการใช้ else กับ for ดังโปรแกรมที่ 6.21 โดยโปรแกรมดังกล่าวจะหาค่าจำนวนเฉพาะ (prime number) คือเลขจำนวนที่ไม่มีเลขของหารมันได้ลงตัว นอกจากตัวของมันเอง และ 1 เช่น 2, 3, 5, 7, 11, 13 และ 17 เป็นต้น

ตัวอย่างโปรแกรมที่ 6.21

Program Example 6.21: else with for & while

```

1 # Testing elase and for loop with prime number
⇒2 for num in range(10, 20): #to iterate between 10 to 20
⇒3     for i in range(2, num): #to iterate on the number
⇒4         if num % i == 0: #to determine the first factor
⇒5             j = num / i #to calculate the second factor
⇒6             print('%d equals %d * %d' % (num, i, j))
⇒7             break
⇒8     else: # else part of the loop
⇒9         print(num, 'is a prime number')

```



OUTPUT

```

10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number

```

จากตัวอย่างโปรแกรมที่ 6.21 บรรทัดที่ 2 คำสั่ง for เริ่มต้นอ่านข้อมูลจำนวนเต็มครั้งละ 1 ค่า จาก range (สร้างช่วงของเลขจำนวนเต็มบางตั้งแต่ 10 ถึง 20) และเก็บไว้ในตัวแปร num ลำดับต่อมา ในบรรทัดที่ 3 เป็นคำสั่ง for ซ้อนอีกชั้นหนึ่ง โดยคำสั่ง for (ชั้นใน) จะนำค่าในตัวแปร num ที่รับมาจาก for ชั้นนอก มาสร้างเป็นช่วงข้อมูลด้วยคำสั่ง range(2, num) ให้กับ for ที่อยู่ชั้นใน ข้อมูลที่สร้างขึ้นจะ ถูกอ่านเข้ามาทำงานที่ล็อก โดยเก็บไว้ในตัวแปร i บรรทัดที่ 4 นำค่าข้อมูลที่อยู่ใน $i \% num$ ผลที่ได้จะ ถูกนำไปตรวจสอบด้วยคำสั่ง if ว่าเท่ากับ 0 หรือไม่ (เป็นการตรวจสอบตัวเลขที่ไม่ใช่ค่า prime number) ถ้าผลลัพธ์มีค่าเท่ากับ 0 แสดงว่าตัวเลขดังกล่าวสามารถหารด้วยตัวเลขใดๆ ลงตัว (ยกเว้นตัวเอง) แสดงว่าไม่ใช่จำนวนเฉพาะ (จำนวนเฉพาะจะหารด้วยตัวเองและ 1 ลงตัวเท่านั้น) ให้โปรแกรมทำการ พิมพ์ตัวเลขที่ไม่ใช่ค่า prime number ดังกล่าวออกทางจอภาพ (บรรทัดที่ 6) และหยุดการทำงานของ for ในรอบนั้นๆ ทันที ด้วยคำสั่ง break (ในบรรทัดที่ 7) แต่ถ้าไม่มีค่า i ได้ๆ ที่ได้จากชุดของข้อมูลที่ สร้างจาก range(2, num) % num ได้ลงตัวเลย ในแต่ละรอบการทำงานของ for ชั้นใน แสดงว่าตัวเลข ดังกล่าวเป็นจำนวนเฉพาะ ดังนั้นโปรแกรมจะทำงานหลังคำสั่ง else (บรรทัดที่ 9) โดยพิมพ์ข้อความว่า 'X is a prime number' (โดย X คือจำนวนเฉพาะ) โปรแกรมจะทำงานวนซ้ำเพื่อค้นหาจำนวนเฉพาะ อย่างนี้ไปเรื่อยๆ จนกว่าค่า num ของ for ลุปนจะมีค่าเท่ากับ 20 โปรแกรมจึงจะยุติการทำงาน

เพื่อให้เห็นภาพการทำงานได้ชัดเจนขึ้น ผู้เขียนจะจำลองการทำงานของโปรแกรมดังกล่าวในรูปที่

6.15 (a) กรณีที่ตัวเลขไม่ใช่จำนวนเฉพาะ, (b) กรณีตัวเลขเป็นจำนวนเฉพาะ

```
for num in range(10,20): [10, 11, 12, 13,..., 19]
    num
        for i in range(2,10): [2, 3, 4, 5, 6, 7, 8, 9]
            if 10 % i == 0 → 10 isn't prime number
```

รูปที่ 6.15 (a) 10 ไม่ใช่จำนวนเฉพาะ เนื่องจาก 10 % ด้วยตัวเลขใดๆ แล้วลงตัว

```
for num in range(10,20): [10, 11, 12, 13,..., 19]
    num
        for i in range(2,11): [2, 3, 4, 5, 6, 7, 8, 9, 10]
            if 11 % i != 0 → 11 is prime number
```

รูปที่ 6.15 (b) 11 คือจำนวนเฉพาะ เนื่องจากไม่มีตัวเลขใด % 11 ลงตัว

ตัวอย่างโปรแกรมที่ 6.22 จะเขียนโปรแกรมเพื่อพิสูจน์ว่าสมการดังต่อไปนี้เป็นจริง

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} -$$

ตัวอย่างอินพุต: Enter integer number (odd) : 15

ตัวอย่างเอาต์พุต: Result of PI/4 = : 0.7543

ตัวอย่างโปรแกรมที่ 6.22

Program Example 6.22: PI/4

```
1 # calculating PI
⇒2 n = int(input("Enter integer number(odd) :"))
⇒3 flag = False      #flag = True (+), flag = False (-)
⇒4 num = 1
⇒5 for i in range(3, n+2, 2):
⇒6     if flag == False:
7         num = num - 1 / i
8         flag = True
⇒9     else:
10        num = num + 1 / i
11        flag = False
⇒12 print ("Result of PI/4 = %.4f" %num)
```



OUTPUT

Enter integer number(odd) :15
Result of PI/4 = 0.7543

จากโปรแกรมตัวอย่างที่ 6.22 บรรทัดที่ 2 โปรแกรมเริ่มต้นอ่านค่าข้อมูลจากแป้นพิมพ์แล้วทำการแปลงสตริงที่รับเข้ามาเป็นเลขจำนวนเต็มด้วยฟังชัน int ผลลัพธ์เก็บไว้ในตัวแปร n บรรทัดที่ 3 ตัวแปร flag จะเป็นตัวแปรที่ใช้สำหรับตรวจสอบว่าเป็นการบวกหรือการลบ เมื่อ flag เท่ากับ False จะเป็นการลบ แต่ถ้า flag เป็น True จะเป็นการบวก บรรทัดที่ 4 ตัวแปร num ใช้สำหรับเก็บค่าผลรวมทั้งหมดโดยกำหนดค่าเริ่มต้นให้เป็น 1 (Operand ตัวแรกของสมการ) บรรทัดที่ 5 คำสั่ง for จะนำค่าข้อมูลที่สร้างโดย range ที่มีช่วงค่าข้อมูลระหว่าง 3 ถึง n+2 (เนื่องจากคำสั่ง range จะสร้างชุดข้อมูลที่น้อยกว่าที่ระบุไว้ 1 ค่า เช่น range(0, 2) ค่าที่ได้คือ 0, 1 การบวกด้วย 2 จะทำให้ได้ข้อมูลครบตามสมการ) และเป็นจำนวนคี่ เช่น 3, 5, 7, 9 เป็นต้น มาเก็บไว้ในตัวแปร i ต่อจากนั้นบรรทัดที่ 6 โปรแกรมจะตรวจสอบค่าในตัวแปร flag ว่ามีค่าเป็น False หรือไม่ ถ้าค่าในตัวแปร flag เป็น False โปรแกรมจะคำนวณด้วยสูตรการลบ เช่น $-\frac{1}{3}, -\frac{1}{7}$ (บรรทัดที่ 7) เป็นต้น แต่ถ้า flag เป็น True โปรแกรมจะคำนวณด้วยสูตรการบวก เช่น $+\frac{1}{5}, +\frac{1}{9}$ (บรรทัดที่ 9) โดยค่า flag จะถูกกำหนดให้สลับค่ากันระหว่างบวกและลบ หลังจากที่มีการคำนวณการบวกหรือลบเสร็จแล้ว (บรรทัดที่ 8 และ 11) เนื่องจากสมการจะคำนวณบวกและลบสลับกันไปเรื่อยๆ จนกว่าจะหมดข้อมูลใน loop for ขั้นตอนสุดท้ายบรรทัดที่ 12 โปรแกรมจะพิมพ์ผลรวมทั้งหมดออกทางจอภาพ โดยกำหนดรูปแบบการแสดงผลเป็นหน่วย 4 ตำแหน่ง (%.4f)

ตัวอย่างโปรแกรมที่ 6.23 โปรแกรม infinite loop แบบไม่รู้จบ (อีกแบบหนึ่ง)

Program Example 6.23: infinite loop

```

1 # Infinite loop
2 x = 1
3 ➔ while True:
4     print("To exit program, please CTRL + c now!" % (x))
5     x += 1

```



OUTPUT

```

To exit program, please CTRL + c now!
...

```

จากโปรแกรมที่ 6.23 เป็นการเขียนโปรแกรมทำซ้ำแบบไม่รู้จบ (อีกแบบหนึ่ง) โดยกำหนดให้ while มีค่าเท่ากับจริงตลอดเวลา (บรรทัดที่ 3) ส่งผลให้โปรแกรมพิมพ์ข้อความออกจอภาพแบบไม่หยุด เมื่อต้องการหยุดโปรแกรมให้กดปุ่ม CTRL + c พร้อมตัวอักษร c ประโยชน์ของโปรแกรมประเภทนี้ใช้สำหรับเป็นฟังชันหลักในการควบคุมการทำงานของฟังชันอื่นๆ เช่น เคอร์แนล (Kernel) ของระบบปฏิบัติการ โปรแกรมหลักในเกมส์คอมพิวเตอร์ หรือโปรแกรมประเภท Client-Server ที่โปรแกรมบนเครื่องเซอร์เวอร์ต้องรอรับการร้องขอการให้บริการตลอดเวลาโดยไม่มีการหยุดโปรแกรม เป็นต้น

ตัวอย่างโปรแกรมที่ 6.24 โปรแกรม for loop กับสตริง

Program Example 6.24: string with for loop

```

1 string = "Hello World"
2 for x in string:
3     print (x)

```


OUTPUT

H
e
l
l
o

W
o
r
l
d

ตัวอย่างโปรแกรมที่ 6.25 โปรแกรม for loop กับลิสต์

Program Example 6.25: list with for loop

```

1 collection = ['hey', 5, 'd']
2 for x in collection:
3     print (x)

```


OUTPUT

hey
5
d

ตัวอย่างโปรแกรมที่ 6.26 โปรแกรม for loop กับลิสต์ซ้อนลิสต์

Program Example 6.26: list[list] with for loop

```

1 list_of_lists = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
2 for list in list_of_lists:
3     for x in list:
4         print (x)

```


OUTPUT

1
2
3
4
5
6
7
8
9

ตัวอย่างโปรแกรมที่ 6.27 โปรแกรม for loop กับ딕ชันนารี

Program Example 6.27: dictionary with for loop

```

1 knights = {'gallahad': 'the pure', 'robin': 'the brave'}
2 for k, v in knights.items():

```

```
3 |     print ("key = %s and value = %s"%(k, v))
```



OUTPUT

```
key = gallahad and value = the pure
key = robin and value = the brave
```

ลูปซ้อน (Nested loops)

ไฟรอนอนุญาตให้ผู้เขียนโปรแกรมสามารถใช้ลูปซ้อนได้ (การวาง for loop ไว้ภายใน for loop)
มีรูปแบบคำสั่งดังนี้

ลูปซ้อนของ for

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statements(s)
        statements(s)
```

ลูปซ้อนของ while

```
while expression:
    while expression:
        statement(s)
        statement(s)
```

ลูปซ้อนของ for กับ while

```
for iterating_var in sequence:
    while expression:
        statement(s)
        statement(s)
```

การวางแผนลูปซ้อนภายในไฟรอนไม่ได้จำกัดว่ามีได้ทั้งหมดกี่ชั้น ดังนั้นผู้เขียนโปรแกรมสามารถวางแผนลูปซ้อนได้ตามอิสระ ขึ้นอยู่กับปัญหาว่ามีความซับซ้อนมากน้อยเพียงใด แต่สิ่งที่ควรจดจำไว้เสมอคือ การวางแผนลูปที่ซ้อนกันมากๆ จะทำให้โปรแกรมอ่านหรือแก้ไขข้อผิดพลาดได้ยากขึ้นด้วย จะเกิดปัญหารื่องของการย่อหน้าภาษาในลูป อาจจะส่งผลให้โปรแกรมทำงานผิดพลาด ตรงจุดนี้ถือว่าเป็นจุดอ่อนของไฟรอนก็ได้ เพราะการไม่มีเครื่องหมายบอกขอบเขต ทำให้บางครั้งต้องใช้สายตาในการหาขอบเขตของคำสั่งเอง พิจารณาจากโปรแกรมต่อไปนี้

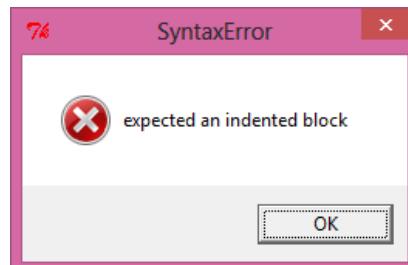
```
for i in range(1, 10):
    print("For loop for i")
    for j in range(1, i):
        print("For loop for j")
```

```

for k in range(1, j):
    print("For loop for k")
print("Good bye!")

```

จากโปรแกรมข้างต้นคำสั่ง `print ("Good bye!")` เป็นคำสั่งของ `for` อันไหนกันแน่ เป็นไปได้ที่จะเขียนโปรแกรมอยู่ มีการกด space bar บ้าง ใช้ปุ่ม tab บ้าง จึงทำให้ย่อหน้าไม่ตรงกันพอดี 既然ในกรณีตัวอย่าง โปรแกรมจะแจ้งเตือนว่า `expected an indented block` แปลว่า มีบางคำสั่งที่อยู่นอกขอบเขต (block) เมื่อกดปุ่ม Ok เครื่องเซอร์จะไปอยู่ในตำแหน่งที่คาดว่าโปรแกรมจะผิดพลาด ผู้เขียนโปรแกรมต้องค้นหาเอาเองว่า ข้อผิดพลาดอยู่ตรงไหน ซึ่งในการนี้ของการเว้นย่อหน้าไม่ตรงกันนี้ ค่อนข้างที่จะค้นหายากพอสมควร



ตัวอย่างโปรแกรมที่ 6.28 โปรแกรมแสดงตารางเวลา

Program Example 6.28: timetable

```

1 # timetable
2 for row in range(1, 10):
3     for col in range(1, 10):
4         prod = row * col
5         if prod < 10:
6             print(' ', end = '')
7         print(row * col, ' ', end = '')
8     print()

```


OUTPUT

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

จากตัวอย่างโปรแกรมที่ 6.28 บรรทัดที่ 6 และ 7 เป็นการใช้คำสั่ง `print` ใน Python 3 ซึ่งจะอนุญาตให้ผู้เขียนโปรแกรมสามารถเพิ่มอักษร หรือข้อความต่อห้ายคำสั่ง แทนการขึ้นบรรทัดใหม่ได้ โดยกำหนดในตัวแปร `end` เช่น เมื่อกำหนด `end = "` โปรแกรมจะไม่พิมพ์ค่าใดๆ และไม่ขึ้นบรรทัดใหม่ สำหรับในบรรทัดที่ 6 คำสั่ง `print(' ', end = '')` โปรแกรมจะพิมพ์อักษรว่าง (' ') และหยุดรอเพื่อพิมพ์

อักษรตัวถัดไป บรรทัดที่ 7 คำสั่ง `print(row * col, ' ', end = "")` โปรแกรมจะพิมพ์ผลลัพธ์ของค่า `row * col` ต่อด้วยอักษรว่าง (' ') และหยุดรอเพื่อพิมพ์อักษรตัวถัดไป (`end = "`)

ตัวอย่างโปรแกรมที่ 6.29 โปรแกรมเชื่อม 2 ลิสต์เข้าด้วยกัน

Program Example 6.29:

```
1 # combination of 2 lists
2 combs = []
3 for x in [1,2,3]:
4     for y in [3,1,4]:
5         if x != y:
6             combs.append((x, y))
7 print(combs)
```



OUTPUT

```
[ (1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4) ]
```

จบบทที่ 6

บทที่ 7

ฟังก์ชัน

(Functions)



1. ฟังก์ชันคืออะไร?

ฟังก์ชัน คือ โปรแกรมย่อยหรืองานย่อยๆ (Sub-program) ภายในโปรแกรมขนาดใหญ่ หรือ บางครั้งเรียกว่า เมธอด (Method) หรือรูทีน (Routine) ก็ได้ โดยปกติโปรแกรมที่มีขนาดใหญ่ จะ ประกอบด้วยคำสั่งต่างๆ มากมายและในโปรแกรมขนาดใหญ่นั้นจะมีคำสั่งที่ทำงานเหมือนกัน ซ้ำกัน หรือถูกเรียกใช้งานอยู่เป็นประจำภายนอก ดังนั้นเพื่อลดคำสั่งให้โปรแกรมสั้นและกระชับลง จึงได้ รวบรวมคำสั่งที่ทำหน้าที่เหมือนกันเหล่านั้นเข้าไว้ด้วยกันเป็นโปรแกรมย่อย เมื่อต้องการเรียกใช้ โปรแกรมย่อยเหล่านั้นตรงจุดใดๆ ในโปรแกรม สามารถเรียกใช้โดยผ่านทางชื่อของฟังก์ชันได้ทันที (Function call) กรณีการเขียนโปรแกรมเชิงโครงสร้าง (Structural programming) นิยมเรียกว่า “ฟังก์ชัน” แต่ในการเขียนโปรแกรมเชิงวัตถุ (Object oriented programming) เรียกว่า “เมธอด” ประเภทของ ฟังก์ชันแบ่งออกเป็น 2 ชนิดคือ ฟังก์ชันมาตรฐาน (Standard functions) ที่มีอยู่ในไลบรารีของภาษาไป รอน และฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้นเอง (User defined functions) ฟังก์ชันที่มีอยู่แล้วในภาษา ไพธอนเมื่อเรียกใช้งานจะต้องทำการ import แฟ้มเข้ามาก่อนเสมอ ในบทนี้จะกล่าวถึงการสร้างฟังก์ชัน ที่ผู้เขียนโปรแกรมสร้างขึ้นใช้งานเอง



Note: ฟังก์ชัน Vs เมธอด แม้ว่าฟังก์ชันและเมธอดจะมีลักษณะการทำงานที่คล้ายกัน แต่ใน การทำงานจริงนั้นจะมีความแตกต่างกันอยู่พอสมควร เนื่องจากแนวคิดโปรแกรมเชิงวัตถุมอง ว่า เมธอดคือพฤติกรรมต่างๆ ของวัตถุ (Object) ที่แสดงออก ซึ่งพฤติกรรมอาจจะไม่ใช้งาน ประจำเหมือนที่ฟังก์ชันทำก็ได้ ซึ่งจะได้ศึกษาและความคิดเชิงวัตถุอย่างละเอียดในบทที่ 11

2. ประโยชน์ของฟังก์ชัน

- ช่วยลดคำสั่งที่ซ้ำซ้อนกันในโปรแกรม

2. ช่วยให้ผู้พัฒนาโปรแกรมสามารถปรับปรุงและแก้ไขโปรแกรมได้อย่างรวดเร็ว เนื่องจาก พังก์ชันแต่ละพังก์ชันนั้นมีหน้าที่ที่ชัดเจนในตัวเอง เช่น `read_input` คือพังก์ชันสำหรับ อ่านค่าอินพุตเข้ามาทำงานในโปรแกรม หรือ `print` ทำหน้าที่พิมพ์ข้อความ
3. ช่วยทำให้โปรแกรมมีความกระหึ่ม ทำให้เข้าใจง่ายและรวดเร็ว เพราะโปรแกรมถูก แบ่งออกตามหน้าที่ของงานชัดเจน
4. พังก์ชันสามารถนำกลับมาใช้ได้อีกหลายครั้ง (reusable code)
5. ป้องกันข้อผิดพลาดได้ดีเพราะงานจะถูกแบ่งตามหน้าที่ชัดเจน การเขียนโปรแกรมจะไม่ ก้าวภายนอกในพังก์ชันอื่นๆ ที่ไม่เกี่ยวข้อง
6. ช่วยให้หาข้อผิดพลาดของโปรแกรมได้รวดเร็วและเป็นระบบ กรณีถ้าโปรแกรมเกิด ข้อผิดพลาดเกิดขึ้นในขณะทำงาน การทดสอบจะทดสอบตามพังก์ชัน
7. พังก์ชันมีการทำงานเป็นอิสระ สามารถนำพังก์ชันที่ถูกสร้างไว้และมีประสิทธิภาพเก็บ ไว้เป็นโมดูล คลาส หรือไลบรารี เพื่อนำไปใช้งานต่อได้ในอนาคตได้

3. การประกาศพังก์ชัน (Defining a function)

ผู้เขียนโปรแกรมสามารถสร้างพังก์ชันขึ้นมาใช้ได้เอง โดยมีกฎการสร้างดังนี้

1. การประกาศพังก์ชันใช้คำว่า `def` นำหน้า ตามด้วยชื่อพังก์ชันและเครื่องหมาย `()`: บีด ท้าย ! เช่น `def myfunc()`: ชื่อของพังก์ชันจะต้องไม่ซ้ำกับคำส่วน ควรสื่อความหมายให้ ตรงกับหน้าที่ของพังก์ชัน
2. กรณีที่พังก์ชันมีพารามิเตอร์ (Parameters) ในพังก์ชัน ให้ใส่พารามิเตอร์เหล่านั้นไว้ใน เครื่องหมาย `()` เช่น `def myfunc(para1, para2)`: พารามิเตอร์สามารถมีได้มากกว่า 1 ตัวแปรได้
3. ให้ชื่อของพังก์ชันให้คำสั่งแรกในพังก์ชัน เป็นคำอธิบายโปรแกรมได้ (Documentation string) โดยที่ไฟล์จะไม่เปลี่ยนความหมาย เช่น

```
def myfunc(para1, para2):
```

"This statement is the documentation string"

Statement(s)

4. คำสั่งในพังก์ชันจะเริ่มต้นหลังเครื่องหมาย :
5. พังก์ชันจะใช้คำสั่ง `return` ในการส่งค่าหรืออัปเดตต่อๆ กันไปยังผู้ที่เรียก กรณีที่ ไม่มีการส่งคืนค่าใดๆ กลับ ไฟล์จะถูกอ่านเป็นการส่งกลับด้วยค่า `None`

รูปแบบการประกาศพังก์ชันดังนี้

```
def functionname( [parameters] ):
    "function_docstring"
    statement(s)
    return [expression]
```

ตัวอย่างการประกาศฟังก์ชันในโปรแกรมที่ 7.1

ตัวอย่างโปรแกรมที่ 7.1 การประกาศฟังก์ชัน

Program Example 7.1: define function

1	# Defining user function
⇒2	def printme(str):
⇒3	"This prints a passed string into this function"
⇒4	print (str)
⇒5	Return

จากตัวอย่างโปรแกรมที่ 7.1 เป็นการประกาศฟังก์ชันอย่างง่าย โดยเริ่มจากบรรทัดที่ 2 ประกาศว่า เป็นฟังก์ชันเมื่อว่า printme ทำหน้าที่พิมพ์ข้อความออกจากภาพและมีพารามิเตอร์สำหรับใช้งานใน ฟังก์ชันนี้ 1 ตัวคือ str บรรทัดที่ 3 เป็นการอธิบายการทำงานของฟังก์ชันนี้ (เพราะจะไม่มีการแปลง ความหมายของบรรทัดนี้) คำสั่งถัดไปในบรรทัดที่ 4 เป็นการเรียกฟังก์ชัน print เพื่อพิมพ์ข้อมูลใน พารามิเตอร์ str ออกจอภาพและคำสั่งสุดท้ายในบรรทัดที่ 5 คือคำสั่งส่งค่ากลับ สำหรับในตัวอย่างนี้จะ ไม่มีการส่งคืนค่าใดๆ กลับไปยังผู้ที่เรียกใช้ แต่ไฟรอนต์ความว่าเป็นค่า None

4. การเรียกใช้ฟังก์ชัน (Calling a function)

เมื่อฟังก์ชันถูกสร้างเสร็จเรียบร้อยแล้ว การเรียกใช้งานสามารถกระทำได้ 3 วิธี คือ เรียกใช้งาน จาก Python shell (prompt), เรียกใช้จากภายในโปรแกรมเดียวกัน และการเรียกใช้จากโปรแกรมอื่นๆ ซึ่งมีรายละเอียดดังนี้

- การเรียกฟังก์ชันจาก Python shell สามารถทำได้โดยเรียกผ่านพร้อมพื้นที่ของ Python shell

โดยตรง เช่น

```
>>> def myfunc():      #Defined function
    print("Testing function")  #Print command
```

```
>>> myfunc()          #Calling function form prompt
Testing function
```

- วิธีเรียกใช้งานจากภายในโปรแกรมเดียวกัน แสดงในโปรแกรมตัวอย่างที่ 7.2 แต่มีข้อห้าม สำหรับวิธีการนี้คือ ห้ามมิให้ผู้เขียนโปรแกรมประกาศฟังก์ชันหลังคำสั่งเรียกใช้งาน เพราะ ไฟรอนจะมองว่ายังไม่มีการประกาศฟังก์ชันไว้ (เป็นจุดอ่อนของการแปลงภาษาแบบ

interpreter ซึ่งแตกต่างจากภาษาซีที่สามารถประกาศฟังก์ชันไว้ที่ใดๆ ก็ได้ เพราะคอมไพล์ของภาษาซีจะแปลทั้งโปรแกรม)



Caution! ห้าม ประกาศฟังก์ชันหลังคำสั่งเรียกใช้งานฟังก์ชัน เพราะจะทำให้โปรแกรมผิดพลาด

ตัวอย่างโปรแกรมที่ 7.2 การเรียกฟังก์ชันภายในโปรแกรมเดียวกัน

Program Example 7.2:

```

1 # Calling user function
⇒2 def printme( str ):
3     "This prints a passed string into this function"
4     print (str)
5     Return
6 # Now you can call printme function
⇒7 printme("I'm first call to user defined function!");
⇒8 printme("Again second call to the same function");

```



I'm first call to user defined function!100

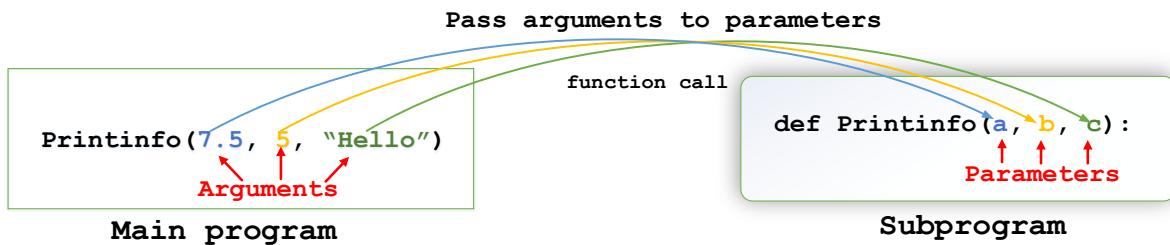
Again second call to the same function

จากโปรแกรมตัวอย่างที่ 7.2 ในบรรทัดที่ 2 โปรแกรมได้ทำการประกาศฟังก์ชันชื่อ printme ทำหน้าที่พิมพ์ข้อความที่รับมาแสดงออกจอภาพ โดยมีพารามิเตอร์ 1 ตัว คือ str ในคำสั่งบรรทัดที่ 7 และ 8 โปรแกรมทำการเรียกฟังก์ชัน printme พร้อมส่งข้อความ "I'm first call to user defined function!" และ "Again second call to the same function" เป็นอาร์กิวเมนต์ให้กับฟังก์ชันเพื่อพิมพ์ข้อความดังกล่าว

3. วิธีเรียกใช้จากโปรแกรมอื่น เป็นวิธีการที่ใช้สำหรับการพัฒนาโปรแกรมขนาดใหญ่และมีความซับซ้อน ซึ่งมักจะนำฟังก์ชันที่เขียนขึ้นมาจัดเก็บเอาไว้เป็นไลบรารีอย่างเป็นระบบ เพื่อต้องการให้โปรแกรมต่างๆ สามารถเรียกใช้งานได้ โดยการนำเข้า (Import) ซึ่งจะอธิบายในหัวข้อโมดูลต่อไป

5. การส่งผ่านอาร์กิวเมนต์ (Pass by reference vs value)

อาร์กิวเมนต์ (Argument) คือ ค่าคงที่หรือค่าของตัวแปรในโปรแกรมหลัก (Main program) ที่ส่งไปเป็นพารามิเตอร์ของโปรแกรมย่อย (Subprogram) เพื่อการคำนวณหรือประมวลผล พารามิเตอร์จะอยู่ภายในวงเล็บหลังชื่อฟังก์ชัน พารามิเตอร์อาจจะมีหรือไม่มีก็ได้ ถ้าไม่มีพารามิเตอร์ให้ใช้เครื่องหมาย () เท่านั้น สำหรับในกรณีที่มีพารามิเตอร์มากกว่าหนึ่งตัว ให้ใช้เครื่องหมายจุลภาค (,) คั่นระหว่างพารามิเตอร์เหล่านั้น การส่งผ่านอาร์กิวเมนต์ในภาษาระดับสูงทั่วไปมี 2 แบบคือ pass by reference และ pass by value แต่สำหรับ Python มีเฉพาะ pass by reference เท่านั้น



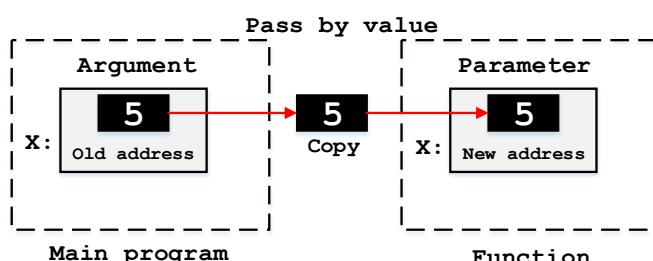
รูปที่ 7.1 Arguments Vs Parameters, Pass arguments to parameters

จากรูปที่ 7.1 แสดงการเปรียบเทียบระหว่าง Argument และ Parameter โดยอาร์กิวเม้นต์คือค่าคงที่ที่ส่งให้กับฟังก์ชันทำงาน แต่พารามิเตอร์คือตัวแปรที่ประกาศไว้ในฟังก์ชันเพื่อรับค่าที่ส่งมาจากอาร์กิวเม้นต์เพื่อใช้ทำงานในฟังก์ชัน (ส่งเป็น Argument รับเป็น Parameter) จำนวนของอาร์กิวเม้นต์จะต้องสัมพันธ์กับจำนวนของพารามิเตอร์ จากตัวอย่างรูปที่ 7.1 อาร์กิวเม้นต์ตัวแรก (7.5) ใน main program จะถูกส่งไปเป็นค่าของพารามิเตอร์ a ในฟังก์ชัน Printinfo (ดังนั้น a จะมีค่าเท่ากับ 7.5 ด้วย) อาร์กิวเม้นต์ตัวที่สอง (5) จะถูกส่งไปเป็นค่าของพารามิเตอร์ b (b มีค่าเท่ากับ 5) และอาร์กิวเม้นต์ตัวที่สาม ("Hello") จะถูกส่งไปเป็นค่าของพารามิเตอร์ c (c มีค่าเท่ากับ "Hello") เป็นต้น

Pass by value คือ การสำเนา (Copy) ค่าข้อมูลจากโปรแกรมที่เรียกใช้ให้กับฟังก์ชันที่ถูกเรียกโดยเก็บค่าที่ทำสำเนาไว้ในตัวแปรชนิดท้องถิ่น (local) ภายในฟังก์ชันที่ถูกเรียกเพื่อประมวลผลหรือกล่าวให้กระชับคือ เป็นการผ่านค่าจริงๆ ของตัวแปรให้ฟังก์ชัน ดังรูปที่ 7.2

ผลการทำงานของ Pass by value คือ

- การแก้ไขหรือเปลี่ยนแปลงข้อมูลในฟังก์ชันจะมีผลเฉพาะในฟังก์ชัน (Local) เท่านั้น จะไม่มีผลใดๆ กับข้อมูลในโปรแกรมที่เรียกใช้
- การส่งคืนค่าจากฟังก์ชันด้วยคำสั่ง return จะเป็นการสำเนาข้อมูลจากภายในฟังก์ชันกลับไปยังผู้เรียกและส่งค่าข้อมูลกลับได้เพียงค่าเดียวเท่านั้น
- ใช้พื้นที่หน่วยความจำในการทำงานมาก เพราะต้องสำเนาข้อมูลทุกพารามิเตอร์ที่ส่งไปยังฟังก์ชัน



รูปที่ 7.2 Pass by value



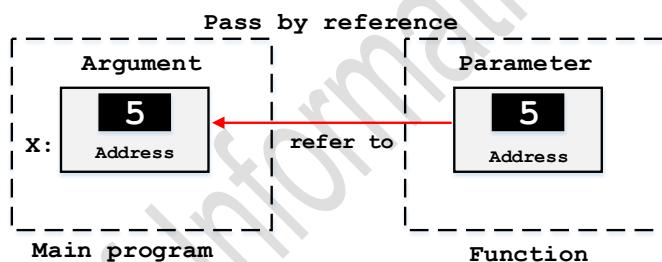
Note: อาร์กิวเมนต์ (Arguments) ใช้ในโปรแกรมหลัก หรือฟังก์ชันที่เรียกใช้งาน

พารามิเตอร์ (Parameters) ใช้ในโปรแกรมย่อย หรือฟังก์ชันที่ถูกเรียกใช้งาน

Pass by reference คือ การส่งที่อยู่ (Address) ของตัวแปรจากโปรแกรมที่เรียกใช้ให้กับฟังก์ชันที่ถูกเรียก หรือพูดง่ายๆ คือ การผ่านที่อยู่ของตัวแปร ดังรูปที่ 7.3

ผลการทำงานของ **Pass by reference** คือ

- การแก้ไขหรือเปลี่ยนแปลงข้อมูลในฟังก์ชันจะมีผลกระทบกับข้อมูลในโปรแกรมที่เรียกใช้ด้วย
- การส่งค่าจากฟังก์ชันด้วยคำสั่ง return จะสามารถส่งกลับได้หลายค่า
- ประหยัดพื้นที่หน่วยความจำ เพราะตัวแปรไม่ได้ถูกคัดลอกเหมือนกับการส่งแบบ pass by value
- ประหยัดเวลา เพราะเข้าถึงหน่วยความจำในตำแหน่งเดียวกัน ทำให้มีจำเป็นต้องคัดลอกตัวแปร



รูปที่ 7.3 Pass by reference



Note: การส่งค่าอาร์กิวเมนต์ในภาษา Python จะมีเฉพาะชนิด Pass by reference เท่านั้น ไม่มีการส่งค่าแบบ Pass by value

ตัวอย่าง และการใช้งาน Pass by reference ดังต่อไปนี้

ตัวอย่างโปรแกรมที่ 7.3 การส่งอาร์กิวเมนต์แบบ Pass by reference

Program Example 7.3: pass by reference

```

⇒1 def changeme(mylist):
2     "This changes a passed list into this function"
3     mylist.append([1,2,3,4]);
4     print ("Values inside the function: ", mylist)
5     return
6 # Now you can call changeme function
7 mylist = [10,20,30];
8 changeme(mylist);
9 print ("Values outside the function: ", mylist)
  
```



OUTPUT

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

จากตัวอย่างโปรแกรมที่ 7.3 บรรทัดที่ 1 ทำการประกาศฟังก์ชันชื่อ `changeme` ทำหน้าที่เชื่อมข้อมูลในลิสต์ 2 ลิสต์เข้าด้วยกัน โดยมีพารามิเตอร์ 1 ตัวคือ `mylist` เพื่อรับค่าข้อมูลจากโปรแกรมที่เรียกใช้งาน ซึ่งเป็นการส่งค่าตัวแปรชนิด `pass by reference` เมื่อรับค่าพารามิเตอร์มาแล้ว โปรแกรมจะนำค่าในพารามิเตอร์ดังกล่าว ซึ่งเป็นชนิดลิสต์ มาเชื่อมกับรายการการลิสต์ที่ฟังก์ชันกำหนดขึ้นคือ `[1, 2, 3, 4]` ด้วยเมธอด `append` (บรรทัดที่ 3) เมื่อเสร็จจากการเชื่อมต่อลิสต์แล้ว บรรทัดที่ 4 โปรแกรมจะทำการพิมพ์ข้อมูลในลิสต์ที่เชื่อมต่อแล้วออกจอภาพ ในบรรทัดสุดท้ายของฟังก์ชัน (บรรทัดที่ 5) จะใช้คำสั่ง `return` เพื่อบอกว่าเป็นการจบฟังก์ชันแล้ว โดยไม่มีการส่งค่าใดๆ กลับไปให้ผู้เรียก

คำสั่งในบรรทัดที่ 7 โปรแกรมทำการประกาศตัวแปรชนิดลิสต์ชื่อ `mylist` มีค่าเท่ากับ `[10, 20, 30]` จากนั้นบรรทัดที่ 8 โปรแกรมทำการเรียกฟังก์ชัน `changeme` พร้อมส่งอาร์กิวเมนต์คือ `mylist` ไปให้ฟังก์ชัน ผลลัพธ์ที่ได้คือ `[10, 20, 30, [1, 2, 3, 4]]` บรรทัดที่ 9 โปรแกรมทำการพิมพ์ข้อมูลใน `mylist` ใหม่อีกครั้ง ผลลัพธ์ที่ได้คือ `[10, 20, 30, [1, 2, 3, 4]]` ซึ่งเหมือนกับผลลัพธ์ที่ได้จากคำสั่งในบรรทัดที่ 8 และงให้เห็นว่าตัวแปร `mylist` ถูกอ้างอิงจากตำแหน่งที่อยู่เดียวกัน เป็นเพราะคุณสมบัติของการใช้ตัวแปรแบบ `Pass by reference` นั่นเอง

การแก้ไขข้อมูลของตัวแปรที่อ้างอิงแบบ `Pass by reference` จะมีผลกระทบทั้งในโปรแกรมที่เรียกใช้และภายในฟังก์ชันที่ถูกเรียก แต่ในกรณีที่ผู้เขียนโปรแกรมทำการประกาศชื่อตัวแปรให้เหมือนกันกับชื่อพารามิเตอร์ในฟังก์ชัน และกำหนดค่าใหม่ให้กับตัวแปรดังกล่าว ไฟรอนจะมองว่าไม่ใช่ตัวแปรเดียวกัน เรียกตัวแปรแบบนี้ว่าตัวแปรห้องถิน (`Local variable`)

ตัวอย่างโปรแกรมที่ 7.4 ตัวแปรชนิดห้องถิน (local variable)

Program Example 7.4: local variable

```

1 # Function definition is here
⇒2 def changeme(mylist):
3     "This changes a passed list into this function"
⇒4     mylist = [1,2,3,4]; # Assign new reference in mylist
⇒5     print ("Values inside the function: ", mylist)
6     return
7 # Now you can call changeme function
⇒8 mylist = [10,20,30];
⇒9 changeme(mylist);
⇒10 print ("Values outside the function: ", mylist)

```



OUTPUT

```
Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]
```

พิจารณาจากโปรแกรมที่ 7.4 บรรทัดที่ 2 โปรแกรมประกาศฟังก์ชันชื่อ `changeme` กำหนดให้กำหนดค่าและพิมพ์ข้อมูล โดยมีพารามิเตอร์ 1 ตัวคือ `mylist` บรรทัดที่ 4 โปรแกรมประกาศตัวแปรชื่อ `mylist` ซึ่งตรงกับชื่อพารามิเตอร์ของฟังก์ชันและกำหนดค่าใหม่ให้กับตัวแปรดังกล่าวเป็น [1, 2, 3, 4] บรรทัดที่ 5 โปรแกรมทำการพิมพ์ค่าข้อมูลที่อยู่ในตัวแปร `mylist` ออกจอภาพ

ในบรรทัดที่ 8 โปรแกรมประกาศตัวแปรชื่อ `mylist` เป็นชนิดลิสต์ และกำหนดค่าเท่ากับ [10, 20, 30] ต่อจากนั้นในบรรทัดที่ 9 โปรแกรมทำการเรียกฟังก์ชัน `changeme` พร้อมส่ง `mylist` เป็นอาร์กิวเมนต์ให้กับฟังก์ชัน ผลลัพธ์ที่พิมพ์จากฟังก์ชันคือ [1, 2, 3, 4] บรรทัดที่ 10 โปรแกรมส่งพิมพ์ค่าข้อมูลที่อยู่ในตัวแปร `mylist` อีกรอบ ผลลัพธ์ที่ได้คือ [10, 20, 30] แสดงให้เห็นว่าตัวแปร `mylist` ที่อยู่ภายนอกและภายในฟังก์ชันไม่ใช่ตัวแปรเดียวกัน



Note: การประกาศตัวแปรเป็นชื่อเดียวกับพารามิเตอร์ที่ประกาศไว้ในฟังก์ชัน ไฟ俏นตีความว่าเป็นตัวแปรคนละตัวกัน

6. ชนิดของอาร์กิวเมนต์ที่ส่งให้ฟังก์ชัน

อาร์กิวเมนต์ที่ส่งให้กับฟังก์ชันแบ่งออกเป็น 4 ประเภทคือ Required arguments, Keyword arguments, Default arguments, Variable-length arguments

Required arguments คือ การส่งจำนวนอาร์กิวเมนต์ให้ตรงกับจำนวนพารามิเตอร์ที่ฟังก์ชันประกาศไว้ โดยอาร์กิวเมนต์สามารถสับเปลี่ยนตำแหน่งกันได้ เนื่องจากไฟ俏นมองทุกอย่างเป็นอ็อปเจกต์ และมีคุณสมบัติการเปลี่ยนชนิดข้อมูลแบบอัตโนมัติ สำหรับตัวอย่างการใช้ required arguments ดังนี้

เมื่อประกาศฟังก์ชัน `myfunc` เป็น

```
def myfunc(number, mylist):
```

โดย `myfunc` มีพารามิเตอร์ 2 ตัวคือ `number` (เป็นชนิดจำนวนเต็ม) และ `mylist` (ชนิดลิสต์) เมื่อเรียกใช้งานฟังก์ชันดังกล่าวจะมีรูปแบบคือ

```
num = 5; mylist = [1, 2, 3, 4]
```

```
myfunc(num, mylist) หรือสับเปลี่ยนของอาร์กิวเมนต์ได้ เช่น myfunc(mylist, num)
```

แต่ถ้าผู้เขียนโปรแกรมเรียกใช้งานฟังก์ชันในลักษณะดังต่อไปนี้จะเกิดข้อผิดพลาดทันที

```
myfunc(num) หรือ myfunc(mylist) หรือ myfunc() ดังตัวอย่างโปรแกรมที่ 7.5
```

ตัวอย่างโปรแกรมที่ 7.5 อาร์กิวเมนต์ชนิด required argument

Program Example 7.5: required argument

```

1 # Required arguments
⇒2 def printme(str):
3     "This prints a passed string into this function"
4     print (str);
5     return;
6 # Now you can call printme function
⇒7 printme();

```


OUTPUT

```

Traceback (most recent call last):
  File "C:/Python33/exam7_4.py", line 8, in <module>
    printme();
TypeError: printme() missing 1 required positional
argument: 'str'

```

จากตัวอย่างโปรแกรมที่ 7.5 โปรแกรมจะเกิดข้อผิดพลาด เพราะว่าฟังก์ชัน printme ต้องการ
พารามิเตอร์ 1 ตัว เป็นชนิดสตริง (บรรทัดที่ 2) แต่เมื่อเรียกใช้งานฟังก์ชันดังกล่าวในบรรทัดที่ 7
โปรแกรมไม่ได้ส่งค่าอาร์กิวเมนต์ตรงตามที่ฟังก์ชันกำหนดไว้

Keyword arguments คือ อาร์กิวเมนต์ที่กำหนดผ่านชื่อตัวแปร เช่น

ประกาศฟังก์ชันเป็น def myfunc(string):

เมื่อเรียกฟังก์ชัน myfunc โดยใช้ Keyword arguments จะมีรูปแบบคือ myfunc(Str ='Testing keyword') ดังโปรแกรมตัวอย่างที่ 7.6

ตัวอย่างโปรแกรมที่ 7.6 อาร์กิวเมนต์ชนิด keyword arguments

Program Example 7.6: keyword arguments

```

1 # Required arguments
⇒2 def printme(str):
3     "This prints a passed string into this function"
4     print (str);
5     return;
6 # Now you can call printme function
⇒7 printme(str = "My string");

```


OUTPUT

My string

จากตัวอย่างโปรแกรมที่ 7.6 บรรทัดที่ 2 โปรแกรมประกาศฟังก์ชันชื่อ printme กำหนดให้พิมพ์
ข้อความออกจอภาพ โดยฟังก์ชันดังกล่าวมีพารามิเตอร์ 1 ตัวคือ str บรรทัดที่ 7 โปรแกรมเรียกฟังก์ชัน
printme พร้อมกับส่งอาร์กิวเมนต์ str = "My string" (เป็นชนิด keyword argument) ให้กับฟังก์ชัน
ผลลัพธ์ที่ได้คือ "My string"

การสลับตำแหน่งของอาร์กิวเมนต์ขณะเรียกใช้ฟังก์ชัน อาจเกิดจากความไม่ได้ตั้งใจของผู้เขียน
โปรแกรม ไฟชอนสามารถแก้ไขความผิดพลาดดังกล่าวได้ โดยการสลับตำแหน่งของตัวแปรเหล่านั้นให้

แต่มีข้อแม้ว่าพารามิเตอร์ในฟังก์ชันกับอาร์กิวเมนต์ที่ส่งจากโปรแกรมที่เรียกต้องเป็นชื่อเดียวกันเท่านั้น ดังโปรแกรมตัวอย่างที่ 7.7

ตัวอย่างโปรแกรมที่ 7.7 การสลับตำแหน่งของอาร์กิวเมนต์

Program Example 7.7: swapping arguments

```

1 # Keyword arguments (swap argument)
2 def printinfo(name, age):
3     "This prints a passed info into this function"
4     print ("Name: ", name);
5     print ("Age ", age);
6     return;
7 # Now you can call printinfo function
8 printinfo(age=50, name="miki")

```



Name: miki
Age 50

จากตัวอย่างโปรแกรมที่ 7.7 บรรทัดที่ 2 โปรแกรมประกาศฟังก์ชันชื่อ printinfo ทำหน้าที่พิมพ์ชื่อและอายุ โดยมีพารามิเตอร์ 2 ตัวคือ name และ age บรรทัดที่ 4 โปรแกรมทำการพิมพ์ชื่อ และบรรทัดที่ 5 พิมพ์อายุ

ในโปรแกรมหลักบรรทัดที่ 8 โปรแกรมเรียกใช้งานฟังก์ชัน printinfo พร้อมกับส่งอาร์กิวเมนต์ไปให้ฟังก์ชัน 2 ตัวคือ age = 50 และ name = miki โดยทดสอบลับที่ตำแหน่งของอาร์กิวเมนต์ไม่ให้ตรงกันกับตำแหน่งของพารามิเตอร์ในฟังก์ชัน printinfo (ในฟังก์ชันกำหนดเป็น name, age) เมื่อฟังก์ชัน printinfo ทำการพิมพ์ค่าของ name และ age ปรากฏว่าผลลัพธ์ที่ได้ถูกต้อง ซึ่งเกิดจากไพร่อนทำการสลับตำแหน่งของพารามิเตอร์ให้อัตโนมัตินั่นเอง

Default arguments คือ อาร์กิวเมนต์ที่ถูกกำหนดค่าไว้ล่วงหน้า เมื่อโปรแกรมที่เรียกใช้ไม่ได้ส่งอาร์กิวเมนต์มาให้กับฟังก์ชันครบตามจำนวนที่ระบุไว้ ฟังก์ชันจะนำค่า default arguments ดังกล่าวมาทำงานแทน ดังตัวอย่างโปรแกรมที่ 7.8

ตัวอย่างโปรแกรมที่ 7.8 default arguments

Program Example 7.8: default arguments

```

1 # Default arguments
2 def printinfo(name, age = 35):
3     "This prints a passed info into this function"
4     print ("Name: ", name);
5     print ("Age: ", age);
6     return;
7 # Now you can call printinfo function
8 printinfo(age=50, name="miki");
9 printinfo(name="miki");

```



Name: miki
Age: 50
Name: miki
Age: 35

จากโปรแกรมที่ 7.8 บรรทัดที่ 2 ทำการประกาศฟังก์ชันชื่อ printinfo ทำหน้าที่พิมพ์ชื่อและอายุออกทางจอภาพ โดยฟังก์ชันดังกล่าวมีพารามิเตอร์ 2 ตัวคือ name และ age สำหรับพารามิเตอร์ age ถูกกำหนดค่าเริ่มไว้คือ age = 35 (เป็นค่า default) บรรทัดที่ 4 เป็นคำสั่งพิมพ์ชื่อและบรรทัดที่ 5 พิมพ์อายุ

บรรทัดที่ 8 โปรแกรมหลักทดสอบเรียกฟังก์ชัน printinfo โดยส่งอาร์กิวเม้นต์มาให้ฟังก์ชันครบตามจำนวน คือ age = 50 และ name ="miki" ผลลัพธ์ที่ได้คือ ฟังก์ชันจะส่งพิมพ์ name = miki และ age = 50 บรรทัดที่ 9 โปรแกรมทำการทดสอบเรียกฟังก์ชัน printinfo อีกรอบ โดยการส่งอาร์กิวเม้นต์ให้ฟังก์ชัน printinfo เพียงตัวเดียวคือ name ="miki" ผลลัพธ์ที่ได้คือ ฟังก์ชันจะพิมพ์ name = miki และ age = 35 (ซึ่ง age = 35 ถูกแทนที่ด้วยค่า default โดยอัตโนมัตินั่นเอง)

Variable-length arguments คือ อาร์กิวเม้นต์ที่สามารถบรรจุข้อมูลไว้ได้แบบไม่จำกัดจำนวน เพื่อใช้ในการณ์ที่ผู้เขียนโปรแกรมไม่ทราบแน่ชัดว่าฟังก์ชันที่เขียนขึ้นจะต้องใช้ข้อมูลในการทำงานกี่ตัว หรืออาจจะมีข้อมูลที่ไม่ได้คาดการณ์ไว้เกิดขึ้นในอนาคต ไฟรอนจึงได้เตรียมอาร์กิวเม้นต์ชนิดนี้ไว้ให้ผู้เขียนโปรแกรมเพื่อความยืดหยุ่นในการทำงาน สำหรับรูปแบบอาร์กิวเม้นต์แบบ Variable-length arguments ดังนี้คือ

```
def functionname([formal_args,] *var_args_tuple):
    "function_docstring"
    function_suite
    return [expression]
```

การประกาศฟังก์ชันโดยใช้อาร์กิวเม้นต์แบบ Variable-length arguments จะมีรูปแบบคล้ายการประกาศฟังก์ชันโดยทั่วไป แต่แตกต่างตรงที่มีสัญลักษณ์ * นำหน้าตัวแปรชนิดทัพเพิลแบบไม่จำกัดจำนวนสมาชิก (คล้ายการประกาศตัวแปรอยู่เครื่องในภาษาซี) สำหรับตัวอย่างการประกาศและใช้งาน อาร์กิวเม้นต์แบบ Variable-length arguments ดังนี้

ตัวอย่างโปรแกรมที่ 7.9 การใช้งาน Variable-length arguments

Program Example 7.9: variable-length arguments

```
1 # Variable-length arguments
2 def printinfo(arg1, *vartuple):
3     "This prints a variable passed arguments"
4     print ("Output of formal arg is : ",arg1)
5     for var in vartuple:
6         print ("Output of virtuple arg is : ",var)
7     return;
```

```

8   | # Now you can call printinfo function
⇒9  | printinfo(10);
⇒10 | printinfo(70, 60, 50);

```



Output of formal arg is : 10
 Output of formal arg is : 70
 Output of virtuple arg is : 60
 Output of virtuple arg is : 50

จากตัวอย่างโปรแกรมที่ 7.9 บรรทัดที่ 2 โปรแกรมประกาศฟังก์ชันชื่อ printinfo ทำหน้าที่พิมพ์ข้อมูล โดยมีพารามิเตอร์ 2 ตัวคือ arg1 (พารามิเตอร์ปกติ) และ *vartuble (เป็นชนิด variable-length สามารถบรรจุข้อมูลหรือสมาชิกได้ไม่จำกัด) บรรทัดที่ 4 ฟังก์ชัน printinfo สั่งพิมพ์ข้อมูลที่อยู่พารามิเตอร์ arg1 ออกจอภาพ บรรทัดที่ 5 โปรแกรมจะอ่านค่าข้อมูลที่เก็บอยู่ในพารามิเตอร์ *vartuble เข้ามาทำงานด้วยคำสั่ง for ข้อมูลที่อ่านได้ในแต่ละครั้งจะถูกพิมพ์ออกจอภาพ (บรรทัดที่ 6) จนกว่าข้อมูลในพารามิเตอร์ *vartuple จะหมด โปรแกรมจึงจะหยุดการทำงานใน for loop

บรรทัดที่ 9 โปรแกรมเรียกใช้ฟังก์ชัน printinfo พร้อมส่งค่าคงที่เป็นอาร์กิวเม้นต์ไปให้ฟังก์ชัน 1 ตัว คือ 10 ผลลัพธ์ที่ได้จากฟังก์ชัน printinfo คือ ข้อความว่า "Output of formal arg is : 10" ซึ่งเพรอนตีความว่าค่า 10 ที่ส่งไปให้ฟังก์ชันเป็นค่าของพารามิเตอร์ arg1 ไม่ใช่พารามิเตอร์ *vartuble

บรรทัดที่ 10 โปรแกรมเรียกฟังก์ชัน printinfo อีกครั้ง ในครั้งนี้กำหนดค่าคงที่เป็นอาร์กิวเม้นต์ให้กับฟังก์ชัน 3 ค่าคือ 70, 60 และ 50 ตามลำดับ เพรอนตีความว่า 70 เป็นค่าของพารามิเตอร์ arg1 และ 60 กับ 50 เป็นค่าของพารามิเตอร์ *vartuple เมื่อฟังก์ชันสั่งพิมพ์ข้อมูล ผลลัพธ์ที่ได้คือ "Output of formal arg is : 70" และ "Output of virtuple arg is : 60, 50" ดังตัวอย่างข้างบน ซึ่งแสดงให้เห็นว่า อาร์กิวเม้นต์แบบ Variable-length arguments สามารถบรรจุข้อมูลได้หลายค่า ซึ่งส่งผลให้การใช้งานมีความยืดหยุ่นมาก

7. การสร้างฟังก์ชันโดยไม่ระบุชื่อ (The Anonymous functions: lambda)



ฟังก์ชันที่ไม่ระบุชื่อ หรือ Lambda expression คือ การสร้างฟังก์ชันที่ผู้เขียนโปรแกรมไม่ต้องการประกาศชื่อฟังก์ชัน แต่ใช้ Keyword "lambda" แทน def ซึ่งถูกใช้ในการประกาศฟังก์ชันแบบปกติ โดยมีเงื่อนไขในการสร้าง lambda ฟังก์ชันดังนี้

1. ฟังก์ชันแบบไม่ระบุชื่อ (Anonymous functions) นี้ สามารถกำหนดจำนวนอาร์กิวเม้นต์ให้กับฟังก์ชันกี่ตัวก็ได้ แต่การส่งค่ากลับ (return) จากฟังก์ชันมิได้เพียงค่าเดียวเท่านั้น
2. ฟังก์ชันแบบไม่ระบุชื่อ ไม่สามารถถูกเรียกใช้เพื่อให้แสดงผล (print) ค่าข้อมูลได้ เพราะจะมีเฉพาะนิพจน์ (Expressions) เท่านั้น

3. ตัวแปรที่ใช้ในการประมวลผลของฟังก์ชันแบบไม่ระบุชื่อ จะอยู่ภายใต้ชื่อของตัวแปร (namespace) ของตัวเองเท่านั้น และไม่สามารถเรียกชื่อ namespace ได้ (แม้ว่าภายในฟังก์ชันจะมีการประกาศตัวแปรแบบ global ไว้ก็ตาม ก็ไม่สามารถเรียกใช้งานได้)
4. เมื่อมีการประกาศฟังก์ชัน def และ lambda เหมือนกัน ไฟรอนจะให้ความสำคัญกับ lambda มากกว่า

ข้อดีของ Anonymous functions

- ไม่ต้องประกาศชื่อฟังก์ชัน แต่ใช้ชื่อเป็นตัวแปรแทน
- คำสั่งมีความกระชับและสั้น เพราะมีเฉพาะนิพจน์เท่านั้น
- มีลักษณะการทำงานคล้ายกับสูตรหรือสมการเฉพาะงาน หรือ inline ในภาษา C/C++
- มีลักษณะการทำงานเป็นแบบ function shorthand หรือ embed a function คือ สามารถฝังฟังก์ชัน lambda เข้าไปในโปรแกรมแต่ละบันทัดได้

ข้อเสียของ Anonymous functions

- ยากต่อการอ่านและทำความเข้าใจ ทำให้นักเขียนโปรแกรมพัฒนาต่อได้ยาก
- ยากต่อการตรวจสอบและแก้ไข เมื่อโปรแกรมเกิดข้อผิดพลาด
- ไม่เหมาะสมกับงานที่มีการประมวลผลที่ซับซ้อนมากๆ เพราะงานขนาดใหญ่ต้องมีการอธิบายการทำงานแต่ละขั้นตอนไว้อย่างละเอียด เพื่อให้ผู้เขียนโปรแกรมสามารถอ่านเข้าใจได้
- การเขียนโปรแกรมโดยใช้ lambda จะเป็นทางการน้อยกว่าการใช้ฟังก์ชัน (def)

Anonymous functions มีรูปแบบการประกาศฟังก์ชัน ดังนี้

lambda [arg1 [,arg2,.....argn]]: expression

โดย lambda เป็นชื่อที่ใช้เรียกแทนชื่อของฟังก์ชัน, arg1, arg2,...,argn เป็นอาร์กิวเม้นต์ที่ส่งให้กับ lambda ทำการประมวลผล และนิพจน์ (expression) คือคำสั่งที่ใช้ประมวลผลข้อมูล, lambda ไม่จำเป็นต้องมีคำสั่ง return ปิดท้ายฟังก์ชัน เมื่อจบคำสั่งสุดท้ายใน lambda ค่าที่ประมวลผลได้จะถูกส่งกลับมาให้แก่ผู้เรียกโดยอัตโนมัติ การเรียกใช้ lambda มีรูปแบบคือ

```
var = lambda arg1, agr2, ... ,argn : expression
print("Test calling lambda : ", var(arg1, arg2,...,argn))
```

โดย var คือตัวแปรที่กำหนดขึ้นเพื่อรับค่าที่ส่งกลับมาจากฟังก์ชัน lambda ตัวอย่างโปรแกรมที่ 7.10 เป็นการสร้างฟังก์ชัน Anonymous functions (lambda) โดยทำหน้าที่รวมค่า 2 จำนวนเข้าด้วยกัน

ตัวอย่างโปรแกรมที่ 7.10

Program Example 7.10: *lambda function*

```
1 # Lambda fuction
2 # Function definition is here
3 sum = lambda arg1, arg2: arg1 + arg2;
4 # Now you can call sum as a function
5 print ("Value of total : ", sum(10, 20))
6 print ("Value of total : ", sum(20, 20))
```



Value of total : 30
Value of total : 40

จากตัวอย่างโปรแกรมที่ 7.10 บรรทัดที่ 3 โปรแกรมทำการสร้างฟังก์ชัน lambda ทำหน้าที่รวมค่า 2 จำนวนเข้าด้วยกัน โดยมีพารามิเตอร์ 2 ตัวคือ arg1 และ arg2 ผลลัพธ์จากการประมวลผลของฟังก์ชัน lambda จะเก็บไว้ในตัวแปรชื่อ sum

บรรทัดที่ 5 โปรแกรมทำการเรียกใช้งานฟังก์ชัน lambda พร้อมกับส่งค่าคงที่เป็นอาร์กิวเม้นต์ให้ฟังก์ชัน lambda 2 ค่าคือ 10 (เก็บไว้ใน arg1) และ 20 (เก็บไว้ใน arg2) ผลลัพธ์ที่ได้จากฟังก์ชัน lambda คือ 30 จากนั้นบรรทัดที่ 6 โปรแกรมเรียกฟังก์ชัน lambda อีกครั้ง พร้อมส่งค่าคงที่เป็นอาร์กิวเม้นต์ให้กับฟังก์ชัน lambda คือ 20 และ 20 ผลลัพธ์ที่ได้คือ 40

ตัวอย่างโปรแกรมที่ 7.11

Program Example 7.11: *lambda vs normal function*

```
1 # Lambda vs normal function in same name
2 def f(x, y, z): return x + y + z
3 f = lambda x, y, z: x - y - z
4 #Calling normal function
5 print("Calling lambda function :", f(2, 2, 2))
6 #calling lambda function
7 print("Calling normal function :", f(2, 3, 4))
```



Calling lambda function : -2
Calling normal function : -5

จากโปรแกรมที่ 7.11 บรรทัดที่ 2 ทำการประกาศฟังก์ชันแบบธรรมด้าชื่อ f ทำหน้าที่หาผลรวมของเลข 3 จำนวนเข้าด้วยกัน โดยพารามิเตอร์ 3 ตัว คือ x, y และ z ตามลำดับ บรรทัดที่ 3 โปรแกรมประกาศฟังก์ชัน lambda ทำหน้าที่หาผลลบของเลข 3 จำนวน โดยมีอาร์กิวเมนต์ 3 ตัวเช่นเดียวกันคือ x, y และ z ตามลำดับ ผลลัพธ์ที่ได้จากการคำนวณจะเก็บไว้ในตัวแปร f

บรรทัดที่ 5 โปรแกรมทำการเรียกใช้งานฟังก์ชัน f พร้อมส่งค่าคงที่เป็นอาร์กิวเมนต์ให้กับฟังก์ชันดังกล่าว 3 ค่าคือ 2, 2 และ 2 ตามลำดับ ผลลัพธ์ที่ได้คือ -2 (ฟังก์ชัน lambda ทำงาน) เนื่องจาก Python ให้ความสำคัญกับฟังก์ชัน lambda หากกว่าฟังก์ชันธรรมด้านั้นเอง แม้ว่าผู้เขียนโปรแกรมตั้งใจจะเรียกฟังก์ชัน f แบบธรรมดาก็ตาม บรรทัดที่ 7 ทำการเรียกฟังก์ชัน f ใหม่อีกรังพร้อมกับส่งค่าคงที่เป็นอาร์กิวเมนต์ให้กับฟังก์ชัน f คือ 2, 3 และ 4 ตามลำดับ ผลลัพธ์ที่ได้คือ -5 ซึ่ง Python จะเรียกใช้งานฟังก์ชัน lambda เมื่อไหร่ก็ได้

ตัวอย่างโปรแกรมที่ 7.12 แสดงการกำหนดค่าเริ่มต้นให้กับพารามิเตอร์ในฟังก์ชัน lambda

Program Example 7.12: lambda vs default value

```

1 # Default value in Lambda function
⇒2 mz = (lambda a = 'A', b = 'B', c = 'C': "[" + a + b + c + "]")
3 #calling lambda function
⇒4 print(mz('A'))
⇒5 print(mz('A', 'B'))
6 print(mz('A', 'B', 'C'))
⇒7 print(mz('A', 'X', 'Y'))
⇒8 print(mz('X', 'Y', 'Z'))

```



OUTPUT

```
[A B C]
[A B C]
[A B C]
[A X Y]
[X Y Z]
```

จากโปรแกรมตัวอย่างที่ 7.12 บรรทัดที่ 2 โปรแกรมทำการประกาศฟังก์ชัน lambda ทำหน้าที่เชื่อมข้อมูลในพารามิเตอร์ a, b และ c เข้าไว้ด้วยกันภายใต้เครื่องหมาย [...] โดยพารามิเตอร์แต่ละตัวจะถูกกำหนดค่าเริ่มต้น (default value) ไว้คือ a = 'A', b = 'B' และ c = 'C' ตามลำดับ ผลลัพธ์ที่ได้จาก การคำนวณในฟังก์ชัน lambda จะเก็บไว้ในตัวแปรชื่อ mz

บรรทัดที่ 4 โปรแกรมทำการเรียกฟังก์ชัน lambda พร้อมกับส่งตัวอักษร 'A' เป็นอาร์กิวเมนต์ เข้าไปด้วย ผลลัพธ์ที่ได้จาก lambda คือ [A B C] แม้ว่าโปรแกรมที่เรียก lambda จะส่งอาร์กิวเมนต์ไปไม่ครบตามจำนวนที่ฟังก์ชัน lambda ได้ประกาศไว้ (lambda ต้องการพารามิเตอร์ 3 ตัว) แต่ Python จะประเมินว่าพารามิเตอร์ที่เหลืออีก 2 ตัว จะใช้ค่า default ที่ฟังก์ชันกำหนดไว้มาทำงานแทน

บรรทัดที่ 5 โปรแกรมเรียกใช้งานฟังก์ชัน lambda เป็นครั้งที่ 2 พร้อมกับตัวอักษร 'A' และ 'B' เป็นอาร์กิวเม้นต์ ซึ่งไม่ครบตามจำนวนที่ฟังก์ชัน lambda ประกาศไว้ เช่นเดิม ผลลัพธ์ที่ได้ยังคงถูกต้อง คือ [A B C] บรรทัดที่ 7 และ 8 โปรแกรมเรียกฟังก์ชัน lambda พร้อมกับส่งตัวอักษรเป็นอาร์กิวเม้นต์ ตามจำนวนที่ฟังก์ชัน lambda กำหนดไว้ แต่เปลี่ยนเป็นอักษรตัวอื่นๆ ที่ไม่ใช่ A, B หรือ C ในที่นี้คือ A, X, Y และ X, Y, Z ตามลำดับ ผลลัพธ์ที่ได้คือ [A X Y] (เนื่องจากฟังก์ชัน lambda จะแทนที่ตัวอักษร A ด้วย A, B ด้วย X และ C ด้วย Y) และ [X Y Z] (ฟังก์ชัน lambda จะแทนที่ตัวอักษร A ด้วย X, B ด้วย Y และ C ด้วย Z) ตามลำดับ

ตัวอย่างโปรแกรมที่ 7.13 แสดงการฝัง lambda เข้าไปในตัวแปรชนิดลิสต์

Program Example 7.13: lambda in list

```

1 # Define lambda in list
2 L = [lambda x: x ** 2, lambda x: x ** 3, lambda x: x ** 4]
3
4 for f in L:
5     print(f(3))
6 print(L[0](11))

```



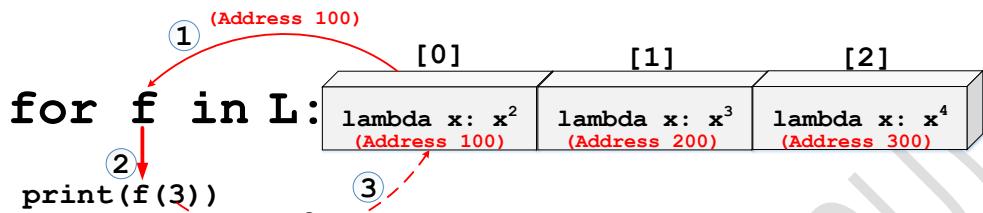
9
27
81
121

โปรแกรมตัวอย่างที่ 7.13 บรรทัดที่ 2 ประมวลฟังก์ชัน lambda ทำหน้าที่คำนวณค่าให้กับ สมาชิกแต่ละตัวในตัวแปรชนิดลิสต์ L (ฝังฟังก์ชัน lambda เข้าไปในสมาชิกของตัวแปรชนิดลิสต์) โดย ฟังก์ชัน lambda มีพารามิเตอร์ 1 ตัว คือ x จากตัวอย่าง สมาชิกในช่องที่ 1 ของลิสต์ L คือ [lambda x: x ** 2] เป็นการหาค่าเลขยกกำลังสองของ x, สมาชิกช่องที่ 2 คือ [lambda x: x ** 3] เป็นการหาค่าเลขยกกำลังสามของ x และสมาชิกช่องที่ 3 คือ [lambda x: x ** 4] คือ การหาค่าเลขยกกำลังสี่ของ x ตามลำดับ

บรรทัดที่ 4 โปรแกรมใช้คำสั่ง for ดึงสมาชิกชนิดอ็อปเจกต์จากตัวแปรลิสต์ L มาทีละค่าเก็บไว้ ในอ็อปเจกต์ f (เรียก f ว่าเป็นอ็อปเจกต์แทนตัวแปรเพราะว่าค่าที่อยู่ใน f เป็นตำแหน่งที่อยู่ของฟังก์ชัน) สำหรับค่าที่เก็บอยู่ในอ็อปเจกต์ f หมายถึงการอ้างถึงฟังก์ชัน lambda x: x ** 2 นั้นเอง (แสดงในรูปที่ 7.4) เมื่อโปรแกรมสั่งพิมพ์ข้อมูลในบรรทัดที่ 5 โดยสั่งค่าคงที่เป็นอาร์กิวเม้นต์ให้กับอ็อปเจกต์ f เท่ากับ 3 สั่งผลให้โปรแกรมเรียกฟังก์ชัน lambda x: x ** 2 มาทำงาน โดยค่า x เท่ากับ 3 ผลลัพธ์ที่ได้คือ $3^2 = 9$ จากนั้นคำสั่ง for (รอบที่ 2) ดึงข้อมูลสมาชิกจากลิสต์ L ในตำแหน่งถัดไปมาทำงาน (สมาชิกตำแหน่งที่ 2) นั่นคือตำแหน่งที่อยู่ของ lambda x: x ** 3 เมื่อโปรแกรมสั่งพิมพ์พร้อมสั่งค่าคงที่เป็นอาร์กิวเม้นต์เท่ากับ 3 ให้กับ f(3) ส่งผลให้เกิดการคำนวณในฟังก์ชัน lambda x: x ** 3 = $3^3 = 27$ คำสั่ง for (รอบที่ 3) ดึงข้อมูลสมาชิกจากลิสต์ L ในตำแหน่งสุดท้ายมาทำงาน นั่นคือตำแหน่งที่อยู่ของ lambda x: x ** 4

จากนั้นเมื่อโปรแกรมสั่งพิมพ์พร้อมส่งค่าคงที่เป็นอาร์กิวเมนต์เท่ากับ 3 ให้กับ $f(3)$ ส่งผลให้เกิดการคำนวณในฟังก์ชัน $\text{lambda } x: x ** 4 = 3^4 = 81$

ในคำสั่งลำดับสุดท้ายบรรทัดที่ 6 โปรแกรมเรียกใช้สมาชิกลำดับที่ 1 ของตัวแปรลิสต์ L ($L[0] = \text{lambda } x: x ** 2$) พร้อมส่งค่าคงที่เป็นอาร์กิวเมนต์ให้กับ $L[0]$ เท่ากับ 11 ผลที่ได้คือ $11^2 = 121$



รูปที่ 7.4 แสดงการฝังฟังก์ชัน lambda ในตัวแปรชนิดลิสต์

8. การส่งค่ากลับจากฟังก์ชัน (The return statement)

คำสั่ง `return` เป็นคำสั่งที่ใช้สำหรับส่งค่ากลับจากฟังก์ชันที่ถูกเรียก โดยค่าที่คืนกลับอาจเป็นค่าที่ถูกประมวลผลในรูปของนิพจน์ ตัวแปร ฟังก์ชัน หรือค่าคงที่ก็ได้ ซึ่งมีรูปแบบคือ

`return [expression]`

ตัวอย่างการส่งค่ากลับที่เกิดจากนิพจน์ทางคณิตศาสตร์ เช่น $\text{return } (33*(s-f)/c*(d+33))$

การส่งค่ากลับจากตัวแปร เช่น $\text{return } y = 2.5 * E^{0.38}$

การส่งค่ากลับจากค่าคงที่ เช่น $\text{return } -1$

การส่งค่ากลับจากฟังก์ชัน เช่น $\text{return } \text{myfunc}(5, 3.5)$

ไม่มีการส่งค่ากลับจากฟังก์ชัน เช่น return หรือ return;

คำสั่ง `return` หรือ `return;` ไฟชอนมองว่าเป็นการส่งคืนค่า `None` ซึ่งไม่สามารถนำค่าดังกล่าวไปใช้ประโยชน์อะไรได้ในการเขียนโปรแกรม สำหรับโปรแกรมตัวอย่างการคืนค่ากลับจากฟังก์ชัน แสดงในโปรแกรมที่ 7.14

ตัวอย่างโปรแกรมที่ 7.14 แสดงการคืนค่าจากฟังก์ชัน

Program Example 7.14: `return`

```

1 # Return from function
2 # Function definition is here
3 def sum(arg1, arg2):
4     # Add both the parameters and return them.
5     total = arg1 + arg2
6     print ("Inside the function : ", total)
7     return total;

```

```

8   # Now you can call sum function
⇒9  total = sum(10, 20);
⇒10 print ("Outside the function : ", total)

```



Inside the function : 30
Outside the function : 30

OUTPUT

จากโปรแกรมที่ 7.14 บรรทัดที่ 3 โปรแกรมประกาศฟังก์ชันชื่อ sum ทำหน้าที่บวกเลข 2

จำนวนเข้าด้วยกันและพิมพ์ค่าที่รวมได้ออกจอภาพ โดยมีพารามิเตอร์ 2 ตัวคือ agr1 และ agr2 บรรทัดที่ 5 โปรแกรมรวมค่าในพารามิเตอร์ arg1 กับ arg2 เข้าไว้ด้วยกัน ผลลัพธ์ที่ได้เก็บไว้ในตัวแปรชื่อ total บรรทัดที่ 6 โปรแกรมสั่งพิมพ์ค่าในตัวแปร total ออกรอภาพ บรรทัดที่ 7 ฟังก์ชันส่งค่าในตัวแปร total กลับไปให้แก่ผู้เรียกใช้งาน

บรรทัดที่ 9 โปรแกรมเรียกใช้ฟังก์ชัน sum พร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์ให้กับฟังก์ชัน คือ 10 และ 20 ผลลัพธ์ที่ได้จากฟังก์ชันคือ 30 บรรทัดที่ 10 โปรแกรมทดสอบพิมพ์ค่าในตัวแปร total ที่ส่งกลับมาจากการฟังก์ชัน sum ผลลัพธ์ที่ได้คือ 30 เมื่อันเดิม จากโปรแกรมข้างต้นมีข้อสังเกตว่าตัวแปร total ที่ประกาศไว้ภายในฟังก์ชันและภายนอกฟังก์ชันไม่ใช่ตัวแปรตัวเดียวกัน

ตัวอย่างโปรแกรมที่ 7.15 เป็นโปรแกรมเครื่องคิดเลขขนาดเล็ก เพื่อสาธิตการสร้างและการเรียกใช้งานฟังก์ชัน ดังนี้

ตัวอย่างโปรแกรมที่ 7.15 โปรแกรมเครื่องคิดเลขขนาดเล็ก

Program Example 7.15: *mini-calculator program*

```

1  # Mini-calculator program
⇒2 def menu():
2  #print what options you have
3  print ("Welcome to calculator program")
4  print ("your options are:")
5  print (" ")
6  print ("(1) Addition")
7  print ("(2) Subtraction")
8  print ("(3) Multiplication")
9  print ("(4) Division")
10 print ("(5) Quit calculator program")
11 print (" ")
12 return int(input("Choose your option: "))

⇒13
13 # this adds two numbers given
⇒16 def add(a, b):
14     print("You chose the Addition")
15     print ("Result of ",a, "+", b, "=", a + b)
16     return a + b

17 # this subtracts two numbers given
⇒22 def sub(a, b):
18     print("You chose the Subtraction")
19
20
21
22
23

```

```

24     print ("Result of ",a, "-", b, "=", a - b)
25     return a - b
26
27 # this multiplies two numbers given
⇒28 def mul(a, b):
28     print("You chose the Multiplication")
29     print ("Result of ",a, "*", b, "=", a * b)
30     return a * b
31
32
33 # this divides two numbers given
⇒34 def div(a, b):
34     print("You chose the Division")
35     if b != 0:
36         print ("Result of ",a, "/", b, "=", a / b)
37         return a / b
38     else:
39         print("Can't divide by zero")
40         return False
41
42
43 # this is main program for control all
⇒44 def main():
44
45     loop = 1
46     choice = 0
⇒47     while loop == 1:
47         choice = menu()
48         if choice == 1:
49             add(int(input("Number 1:")),int(input("Number
50             2:")))
⇒51             elif choice == 2:
51                 sub(int(input("Number 1:")),int(input("Number
52                 2:")))
⇒53             elif choice == 3:
53                 mul(int(input("Number 1:")),int(input("Number
54                 2:")))
⇒55             elif choice == 4:
55                 div(int(input("Number 1:")),int(input("Number
56                 2:")))
⇒57             elif choice == 5:
57                 loop = 0
⇒59         else:
59             print("Good bye! ")
60
61
⇒62 if __name__ == "__main__":
62     main()
⇒63

```



OUTPUT

Welcome to calculator program

your options are:

- 1) Addition
- 2) Subtraction
- 3) Multiplication
- 4) Division
- 5) Quit calculator program

```

Choose your option: 1
Number 1:5
Number 2:8
You chose the Addition
Result of 5 + 8 = 13
Choose your option: 5
Good bye!

```

จากตัวอย่างโปรแกรมที่ 7.15 เป็นโปรแกรมเครื่องคิดเลขขนาดเล็กที่มีฟังก์ชันการทำงานทั้งหมด 6 ฟังก์ชันคือ ฟังก์ชันหลัก (main), สร้างเมนู (menu), การบวก (add), การลบ (sub), การคูณ (mul), การหาร (div) ดังนี้

บรรทัดที่ 2 ประกาศฟังก์ชันชื่อว่า menu ทำหน้าที่แสดงคุณสมบัติต่างๆ (Features) ของเครื่องคิดเลขที่เตรียมไว้ให้ใช้งาน เช่น เลือก 1 คือการบวก, 2 คือการลบ ถ้าต้องการออกจากโปรแกรมให้เลือก 5 เป็นต้น ฟังก์ชันนี้ไม่ต้องการพารามิเตอร์เพื่อใช้ในการทำงาน แต่ฟังก์ชันจะส่งค่ากลับไปให้แก่ผู้เรียกใช้เป็นเลขจำนวนเต็มที่ผู้ใช้เลือกว่าต้องการให้เครื่องคิดเลขทำอะไร (บรรทัดที่ 13) เช่น ถ้าเลือก 1 คือการบวก เป็นต้น

บรรทัดที่ 16 โปรแกรมประกาศฟังก์ชัน add ทำหน้าที่บวกเลข 2 จำนวนเข้าด้วยกัน และส่งผลลัพธ์ที่ได้จากการบวกคืนให้แก่ผู้เรียก โดยฟังก์ชันมีพารามิเตอร์ 2 ตัวคือ a และ b

บรรทัดที่ 22 โปรแกรมประกาศฟังก์ชัน sub ทำหน้าที่ลบเลข 2 จำนวน และส่งผลลัพธ์ที่ได้จากการลบคืนให้แก่ผู้เรียก โดยฟังก์ชันมีพารามิเตอร์ 2 ตัวคือ a และ b

บรรทัดที่ 28 โปรแกรมประกาศฟังก์ชัน mul ทำหน้าที่คูณเลข 2 จำนวน และส่งผลลัพธ์ที่ได้จากการคูณคืนให้แก่ผู้เรียก โดยฟังก์ชันมีพารามิเตอร์ 2 ตัวคือ a และ b

บรรทัดที่ 34 โปรแกรมประกาศฟังก์ชัน div ทำหน้าที่หารเลข 2 จำนวน และส่งผลลัพธ์ที่ได้จากการหารคืนให้แก่ผู้เรียก โดยฟังก์ชันมีพารามิเตอร์ 2 ตัวคือ a และ b ภายในฟังก์ชันมีการตรวจสอบด้วยว่าค่าที่ใช้หารต้องไม่เป็น 0 ถ้าเป็น 0 ฟังก์ชันจะส่งค่ากลับเป็น False

บรรทัดที่ 44 โปรแกรมประกาศฟังก์ชันชื่อ main ทำหน้าที่ควบคุมการทำงานของเครื่องคิดเลข ในฟังก์ชันไม่มีพารามิเตอร์เพื่อใช้ทำงาน บรรทัดที่ 45 โปรแกรมประกาศตัวแปรชื่อว่า loop มีค่าเริ่มต้นเท่ากับ 1 ใช้สำหรับตรวจสอบว่าผู้ใช้ต้องการออกจากโปรแกรมหรือไม่ ถ้า loop เท่ากับ 0 แสดงว่าผู้ใช้ต้องการออกจากโปรแกรม บรรทัดที่ 46 ประกาศตัวแปรชื่อ choice มีค่าเริ่มต้นเป็น 0 มีหน้าที่เก็บค่าตัวเลขจำนวนเต็มที่ส่งกลับมาจากฟังก์ชัน menu (เป็นตัวเลขที่บอกให้โปรแกรมทราบว่าผู้ใช้เลือกใช้คุณสมบัติใดของเครื่องคิดเลข) บรรทัดที่ 47 โปรแกรมตรวจสอบเงื่อนไขด้วยคำสั่ง while ว่า loop มีค่าเท่ากับ 1 หรือไม่ ผลจากการตรวจสอบเป็นจริง (เพราะบรรทัดที่ 45 มีการกำหนดให้ loop = 1) บรรทัดที่ 48 โปรแกรมเรียกฟังก์ชัน menu เพื่อให้ผู้ใช้เลือกว่าต้องการให้เครื่องคิดเลขทำฟังก์ชันอะไร ผลลัพธ์

ที่ส่งกลับมาจากการพิมพ์ชั้น menu คือตัวเลขจำนวนเต็มตั้งแต่ 1 – 5 เท่านั้น แต่ถ้าผู้ใช้ป้อนตัวเลขที่ไม่ใช่ 1 – 5 โปรแกรมจะกลับไปรับค่าใหม่

ถ้าผู้ใช้เลือก 1 (บรรทัดที่ 49) โปรแกรมจะแสดงข้อความว่า "Number 1:" และ "Number 2:" เพื่อรอให้ผู้ใช้ป้อนตัวเลขใดๆ 2 จำนวนทางแป้นพิมพ์ เมื่อผู้ใช้ป้อนข้อมูลทั้ง 2 จำนวนแล้ว โปรแกรมจะไปทำงานในฟังก์ชัน add (การบวก) ผลลัพธ์ที่ได้จะถูกพิมพ์ออกจากการ และโปรแกรมจะกลับมารับคำสั่งต่อไปในโปรแกรมหลัก (main)

ถ้าผู้ใช้เลือก 2 (บรรทัดที่ 51) โปรแกรมจะแสดงข้อความว่า "Number 1:" และ "Number 2:" เช่นเดิม เพื่อรอให้ผู้ใช้ป้อนตัวเลขใดๆ 2 จำนวนทางแป้นพิมพ์ เมื่อผู้ใช้ป้อนข้อมูลทั้ง 2 จำนวนแล้ว โปรแกรมจะไปทำงานในฟังก์ชัน sub (การลบ) ผลลัพธ์ที่ได้จะถูกพิมพ์ออกจากการ และโปรแกรมจะกลับมารับคำสั่งต่อไปในโปรแกรมหลัก (main)

ถ้าผู้ใช้เลือก 3 หรือ 4 (บรรทัดที่ 53 หรือ 55) โปรแกรมจะแสดงข้อความว่า "Number 1:" และ "Number 2:" เช่นเดียวกัน เพื่อรอให้ผู้ใช้ป้อนตัวเลขใดๆ 2 จำนวนทางแป้นพิมพ์ เมื่อผู้ใช้ป้อนข้อมูลทั้ง 2 จำนวนแล้ว โปรแกรมจะไปทำงานในฟังก์ชัน mul (การคูณ) หรือ div (การหาร) ผลลัพธ์ที่ได้จะถูกพิมพ์ออกจากการ และโปรแกรมจะกลับมารับคำสั่งต่อไปในโปรแกรมหลัก (main) เช่นเดิม

ถ้าผู้ใช้เลือก 5 (บรรทัดที่ 57) โปรแกรมจะพิมพ์ข้อความว่า "Good bye!" พร้อมกับยุติการทำงานของโปรแกรมทันที แต่ถ้าผู้ใช้ป้อนเป็นตัวเลขใดๆ ที่ไม่ใช่ 1 – 5 โปรแกรมจะกลับไปรอให้ผู้ใช้ป้อนข้อมูลใหม่ สำหรับເອົາຕີພຸດຂອງການໃຊ້ຈຳປັດໂປຣແກຣມເຄື່ອງຄິດເລີຂະແສດງໃນຕົວອ່າງຂັງບນ

หมายเหตุ: โปรแกรมนี้ใช้ได้กับเลขจำนวนเต็มเท่านั้น ไม่สามารถใช้ได้กับเลขจำนวนจริง ถ้าผู้เขียนต้องการให้โปรแกรมสามารถใช้งานได้ทั้งจำนวนเต็มและจำนวนจริงให้แก้ไขโปรแกรมในบรรทัดที่ 50, 52, 54 และ 56 ดังนี้

จาก `add(int(input("Number 1:")), int(input("Number 2:")))`

เป็น `add(float(input("Number 1:")), float(input("Number 2:")))`

9. การส่งค่ากลับจากฟังก์ชันหลายค่า (Returning multiple values)

ในภาษาจะดับสูงส่วนใหญ่ ฟังก์ชันสามารถส่งค่ากลับได้เพียงค่าเดียวเท่านั้น แต่ใน Python ยอมให้ผู้เขียนโปรแกรมสามารถส่งค่ากลับจากฟังก์ชันได้มากกว่า 1 ค่า โดยอาศัยตัวแปรชนิดทัพเพิลซ้ายในการทำงาน ซึ่งมีรูปแบบการใช้งาน ดังตัวอย่างโปรแกรมที่ 7.16

ตัวอย่างโปรแกรมที่ 7.16 การส่งค่ากลับจากฟังก์ชันหลายค่า

Program Example 7.16: returning multiple values

```

1 # Returning multiple values
2 import random
3 def rollDice():
4     return (1 + random.randrange(6), 1 +
5         random.randrange(6))
6 d1, d2 = rollDice()
7 print (d1,",",d2)
8 d1, d2 = rollDice()
9 print (d1,",",d2)
10 d1, d2 = rollDice()
11 print (d1,",",d2)

```



4 , 3
5 , 3
3 , 5



จากโปรแกรมตัวอย่างที่ 7.16 บรรทัดที่ 3 ประกาศพังก์ชันชื่อ rollDice มีหน้าที่สุ่มแต้มของลูกเต๋า 2 ลูก (บรรทัดที่ 4) ตัวเลขที่สุ่มได้จะอยู่ในช่วงระหว่าง 1 – 6 (เท่ากับจุดบนหน้าของลูกเต๋า) และส่งแต้มที่สุ่มได้ทั้ง 2 ค่ากลับคืนให้กับผู้เรียก

บรรทัดที่ 5 โปรแกรมทำการเรียกใช้พังก์ชัน rollDice ค่าที่ส่งกลับมาจากการพังก์ชัน rollDice มี 2 ค่า เพราะลูกเต๋ามี 2 ลูก (เป็นข้อมูลชนิดทัพเพิล) ค่าทั้งสองจะถูกเก็บไว้ในตัวแปรชนิด int ชื่อ d1 และ d2 ตามลำดับ จากนั้นบรรทัดที่ 6 โปรแกรมทำการพิมพ์ค่าที่เก็บอยู่ในตัวแปร d1 และ d2 สำหรับโปรแกรมบรรทัดที่ 7 – 10 โปรแกรมทำการสุ่มโยนลูกเต๋าหลายๆ ครั้ง

10. ขอบเขตของตัวแปร (Scope of variables)

ขอบเขตของตัวแปรคือ บริเวณหรือสถานที่สามารถเข้าถึงตัวแปรหรืออ้างถึงเพื่อใช้งานได้โดยตัวแปรที่อยู่ในโปรแกรมหลัก เรียกว่า ตัวแปรชนิดโกลบอล (Global) สำหรับตัวแปรที่อยู่ในพังก์ชัน เรียกว่า ตัวแปรชนิดโอลโคล (Local) ซึ่งมีรายละเอียดดังต่อไปนี้

- **ตัวแปรโกลบอล** ตัวแปรชนิดนี้จะประกาศไว้ในส่วนของโปรแกรมหลัก (main program) และสามารถเข้าถึงตัวแปรดังกล่าว ได้ตลอดทั้งโปรแกรม (โปรแกรมส่วนอื่นๆ สามารถเรียกใช้ตัวแปรแบบโกลบอลจากที่ใดๆ ก็ได้ในโปรแกรม)

- ตัวแปรชนิดโลคอล ตัวแปรชนิดนี้จะประกาศไว้ในฟังก์ชันเท่านั้น ขอบเขตการทำงานของตัวแปรจะอยู่เฉพาะภายในฟังก์ชันเท่านั้น เมื่อเสร็จสิ้นการทำงานภายในฟังก์ชัน ตัวแปรเหล่านั้นจะถูกลบทิ้งทันที คุณสมบัติการใช้งานระหว่างตัวแปรโกลบอลและโลคอล สามารถสรุปได้ดังนี้
 - กรณีที่มีการตั้งชื่อตัวแปรชนิดโกลบอลและโลคอลต่างกัน ไฟรอนถือว่าตัวแปรโลคอลจะมีขอบเขตการทำงานเฉพาะภายในฟังก์ชันเท่านั้น โปรแกรมภายนอกฟังก์ชันไม่สามารถเรียกใช้งานตัวแปรโลคอลได้ แต่ตัวแปรโกลบอลสามารถถูกเรียกใช้ได้จากทุกๆ ที่ในโปรแกรม
 - กรณีที่มีการตั้งชื่อตัวแปรชนิดโกลบอลและโลคอลเหมือนกัน เมื่อเรียกใช้งานตัวแปรภายในโปรแกรมหลัก ตัวแปรที่ถูกเรียกใช้คือ ตัวแปรโกลบอล แต่ถ้าเรียกใช้งานตัวแปรภายในฟังก์ชัน ตัวแปรที่ถูกเรียกใช้ คือ ตัวแปรโลคอล
 - กรณีเรียกใช้ตัวแปรภายในฟังก์ชัน แต่ไม่มีการประกาศตัวแปรดังกล่าวไว้ในฟังก์ชัน โปรแกรมจะเรียกใช้ตัวแปรโกลบอลแทน (แต่ต้องประกาศว่าเป็นตัวแปรแบบโกลบอลไว้ด้วย โดยใช้ keyword global มีฉะนั้นจะเกิดข้อผิดพลาด)
 - กรณีเรียกใช้ตัวแปรระหว่างฟังก์ชัน ไม่สามารถทำได้ แต่สามารถแก้ไขได้โดย ประกาศตัวแปรที่ต้องการใช้ระหว่างฟังก์ชันเป็นตัวแปรชนิดโกลบอลแทน

ตัวอย่างโปรแกรมที่ 7.17 แสดงตัวอย่างการใช้งานตัวแปรแบบโกลบอล และโลคอล

Program Example 7.17: Global vs local variables

```

1 # Scope of variables
⇒2 total = 0; # This is global variable.
⇒3 def sum(arg1, arg2):
⇒4     # Add both the parameters and return them.
⇒5     total = arg1 + arg2; # Here total is local variable.
⇒6     print ("Total of local variable inside function : ",
total)
7     return total;
8 # calling sum function
⇒9 sum(5, 10);
⇒10 print ("Total of global variable outside function : ",
total)
  
```



Total of local variable inside function : 15
 Total of global variable outside function : 0

จากโปรแกรมตัวอย่างที่ 7.16 บรรทัดที่ 2 โปรแกรมประกาศตัวแปรโกลบลชื่อว่า total มีค่าเริ่มต้นเท่ากับ 0 บรรทัดที่ 3 ประกาศฟังก์ชันชื่อว่า sum ทำหน้าที่บวกเลข 2 จำนวนเข้าด้วยกัน และส่ง

ผลลัพธ์ที่บวกได้กลับไปยังผู้เรียก โดยฟังก์ชันมีพารามิเตอร์ที่ใช้งาน 2 ตัวคือ arg1 และ arg2 บรรทัดที่ 5 เป็นการบวกระหว่าง arg1 กับ agg2 ผลลัพธ์เก็บไว้ในตัวแปร total บรรทัดที่ 6 โปรแกรมสั่งพิมพ์ผลลัพธ์ออกจากภาพ และบรรทัดที่ 7 ฟังก์ชันส่งค่า total กลับคืนไปยังผู้เรียกใช้งาน

บรรทัดที่ 9 โปรแกรมเรียกใช้ฟังก์ชัน sum พร้อมกับส่งค่าคงที่เป็นอาร์กิวเมนต์ให้กับฟังก์ชัน 2 ค่าคือ 5 และ 10 ตามลำดับ ผลลัพธ์ที่ส่งกลับมาจากฟังก์ชัน sum มีค่าเท่ากับ 15 บรรทัดที่ 10 โปรแกรมทำการพิมพ์ค่าที่อยู่ในตัวแปร total อีกรัง ผลปรากฏว่ามีค่าเท่ากับ 0 มีข้อสังเกตว่าตัวแปร total ภายนอกฟังก์ชันและภายในฟังก์ชันไม่ใช้ตัวเดียวกัน สำหรับตัวแปร total ภายในฟังก์ชันเมื่อโปรแกรมทำงานเสร็จโปรแกรมจะลบตัวแปรดังกล่าวทิ้งทันทีและตัวแปรภายในฟังก์ชันจะมีขอบเขตการทำงานเฉพาะภายในฟังก์ชันเท่านั้น

ตัวอย่างโปรแกรมที่ 7.18 แสดงการกำหนดตัวแปรโกลบอลและโลคอล

Program Example 7.18: **define variables**

```

1      # Define variables
⇒2    def modifyVar1():
3        var1, var2 = 10, 20 #These are local variables
4        print("Local variables are var1 = %d, var2 = %d"%(var1,
var2))
⇒5    def modifyVar2(var1, var2):
6        var1 = 30 #var1 is local, and var2 is global
7        print("Local variable var1 = %d, and global var2 =
%d"%(var1, var2))
⇒8    def modifyVar3(var1):
9        global var2
10       var2 = 3
11       return var1**var2
⇒12   def modifyVar4():
13       global var1
14       var1 = var1**2
15       return
⇒16   var1, var2 = 5, 5
⇒17   modifyVar1()
⇒18   modifyVar2(5, 10)
⇒19   print("Var1 and Var2 in main program = %d, %d"%(var1, var2))
⇒20   print(modifyVar3(var1))
⇒21   print("Var1 and Var2 in main program = %d, %d"%(var1, var2))
⇒22   modifyVar4()
⇒23   print("Var1 and Var2 in main program = %d, %d"%(var1, var2))

```



Local variables are var1 = 10, var2 = 20
 Local variable var1 = 30, and global var2 = 10
 Var1 and Var2 in main program = 5, 5
 125
 Var1 and Var2 in main program = 5, 3
 Var1 and Var2 in main program = 25, 3

จากโปรแกรมที่ 7.17 บรรทัดที่ 2 โปรแกรมประกาศฟังก์ชันชื่อ modifyVar1 มีหน้าที่กำหนดค่าให้กับตัวแปรโลคอล var1 = 10 และ var2 = 20 (บรรทัดที่ 3) และพิมพ์ผลลัพธ์ของตัวแปร var1 และ var2 ออกจากภาพ (บรรทัดที่ 4) ฟังก์ชันนี้ไม่มีพารามิเตอร์เพื่อใช้งาน บรรทัดที่ 5 โปรแกรมประกาศฟังก์ชันชื่อ modifyVar2 มีหน้าที่กำหนดค่าให้กับตัวแปรโลคอล var1 = 20 (บรรทัดที่ 6) และพิมพ์ผลลัพธ์ของตัวแปร var1 และ var2 (โดยใช้ตัวแปร var2 จากภายในนอกฟังก์ชัน) ออกจากภาพ (บรรทัดที่ 7) ฟังก์ชันนี้มีพารามิเตอร์ 2 ตัวคือ var1 และ var2

บรรทัดที่ 8 โปรแกรมประกาศฟังก์ชันชื่อ modifyVar3 มีหน้าที่กำหนดค่าให้กับตัวแปรโกลบอล (โดยใช้ keyword ว่า global นำหน้าตัวแปร ในบรรทัดที่ 9) var2 = 3 (บรรทัดที่ 10) และส่งค่ากลับเป็น var1**var2 กลับไปยังผู้เรียก (บรรทัดที่ 11) บรรทัดที่ 12 โปรแกรมประกาศฟังก์ชันชื่อ modifyVar4 มีหน้าที่กำหนดค่าให้กับตัวแปรโกลบอล var2 = var1**2 (บรรทัดที่ 14) โดยไม่ส่งค่ากลับไปยังผู้เรียก

บรรทัดที่ 16 โปรแกรมประกาศตัวแปรชื่อ var1 มีค่าเท่ากับ 5 และ var2 มีค่าเท่ากับ 5 เช่นเดียวกัน จากนั้นบรรทัดที่ 17 โปรแกรมเรียกฟังก์ชัน modifyVar1 ผลลัพธ์ที่ได้คือ var1 = 10 และ var2 เท่ากับ 20 เพราะฟังก์ชันเรียกใช้ตัวแปรแบบ var1 และ var2 จำกภายในฟังก์ชัน (Local)

บรรทัดที่ 18 โปรแกรมเรียกฟังก์ชัน modifyVar2 พร้อมกับส่งค่าคงที่เป็นอาร์กิวเม้นต์ให้กับฟังก์ชัน 2 ค่าคือ 5 และ 10 ผลลัพธ์ที่ได้คือ var1 = 30 และ var2 เท่ากับ 10 เพราะภายในฟังก์ชันได้กำหนดค่าให้กับตัวแปร var1 = 30 และ var2 เป็นค่าที่รับมาจากภายใน สรุปว่าฟังก์ชันดังกล่าวเรียกใช้ตัวแปรแบบโลคอลทั้ง var1 และ var2 โดยที่ตัวแปร var1 และ var2 ภายนอกฟังก์ชันไม่มีการเปลี่ยนแปลงค่าใดๆ (บรรทัดที่ 19)

บรรทัดที่ 20 โปรแกรมเรียกฟังก์ชัน modifyVar3 พร้อมกับส่งค่าคงที่เป็นอาร์กิวเม้นต์ให้กับฟังก์ชัน 1 ค่าคือ ค่าในตัวแปร var1 ที่อยู่ภายนอกฟังก์ชัน ผลลัพธ์ที่ได้คือ 125 (ค่า 125 เกิดจาก var1**var2 = 5³) สรุปว่าฟังก์ชันดังกล่าวเรียกใช้ตัวแปรแบบโกลบอลทั้ง var1 และ var2 บรรทัดที่ 21 ทดสอบสิ่งพิมพ์ค่าในตัวแปร var1 และ var2 อีกรอบ ผลปรากฏว่าค่าในตัวแปร var2 เปลี่ยนจาก 5 เป็น 3 เพราะฟังก์ชัน modofyVar3 ทำการกำหนดค่าใหม่ให้กับตัวแปรโกลบอลคือ var2 (var2 = 3)

บรรทัดที่ 22 โปรแกรมเรียกฟังก์ชัน modifyVar4 โดยไม่มีการส่งอาร์กิวเม้นต์ใดๆ ให้กับฟังก์ชันซึ่งฟังก์ชันดังกล่าวไม่มีการส่งค่ากลับเพียงแต่กำหนดค่าตัวแปร var1 = va1**2 เท่านั้น จากนั้นบรรทัดที่ 23 โปรแกรมทดสอบพิมพ์ค่า var1 และ var2 อีกรอบ ผลลัพธ์ที่ได้คือ var1 เท่ากับ 25 และ var2 เท่ากับ 3 สรุปว่าฟังก์ชันนี้เรียกใช้ตัวแปรแบบโกลบอลทั้ง var1 และ var2



Note: เมื่อโปรแกรมภายในฟังก์ชันเรียกใช้ตัวแปรจากภายนอกฟังก์ชัน ต้องใช้คำสั่ง global นำหน้าตัวแปรที่ใช้งานอยู่ภายในฟังก์ชัน เช่น **global var1**

11. การเรียกตัวเอง หรือการเวียนเกิด (Recursion)

การเรียกตัวเอง คือ วิธีการแก้ปัญหาแบบหนึ่ง ที่ใช้วิธีเรียกตัวเองซ้ำและทำตามหลักการเดิมไปเรื่อยๆ จนกว่าจะได้คำตอบที่ต้องการ สำหรับปัญหาที่เหมาะสมที่จะใช้การเรียกตัวเองคือ ปัญหาที่สามารถแบ่งย่อยออกเป็นส่วนๆ ได้ เมื่อผ่านกรรมวิธีแก้ปัญหาแบบเรียกตัวเองแล้ว ปัญหาจะถูกแบ่งออกเป็นส่วนย่อยๆ จำนวนมาก ซ้อนกันเป็นชั้นๆ แต่ละชั้นใช้วิธีการแก้ปัญหาแบบเดียวกัน จนกระทั่งถูกแบ่งเป็นปัญหาย่อยที่เล็กที่สุดที่สามารถหาคำตอบได้ (ซึ่งส่วนใหญ่เป็นค่าคงที่) ค่าคงที่ดังกล่าวจะเป็นค่าที่ทำให้สามารถหาคำตอบแบบบันทึกกลับได้ โดยการรวมผลลัพธ์ที่แก้ปัญหาแล้วจากปัญหาย่อยๆ กลับมาเป็นคำตอบของปัญหาทั้งหมด การประภาคพั้งก์ชันการเรียกตัวเองมีรูปแบบดังต่อไปนี้

```
def name([arg1, arg2,...,argn], n):
    statement(s)
    if condition(n):
        return constant
    else:
        return name([arg1, arg2,...,argn], n-1)
```

name คือ ชื่อของฟังก์ชันนิดเรียกตัวเอง arg1, arg2, ..,argn คือพารามิเตอร์ที่ส่งให้กับฟังก์ชันเพื่อใช้ในการประมวลผล (ซึ่งจะมีหรือไม่มีก็ได้ เพราะอยู่ในเครื่องหมาย [...]), n คือ พารามิเตอร์ที่ใช้เป็นเงื่อนไขในการหาค่าที่เล็กที่สุด, n-1 คือ การเรียกซ้ำตัวเองโดยการแบ่งย่อยปัญหาไปเรื่อยๆ จนกว่าเงื่อนไขใน if condition(n) จะเป็นจริง

ตัวอย่างการหาค่ายกกำลัง exponent (exp) ซึ่งเป็นปัญหาที่สามารถแบ่งย่อยเป็นส่วนๆ ได้

$$\text{จากสมการของ } \exp(a^n) = a_1 * a_2 * \dots * a_n$$

$$\text{สมมุติว่า } a = 2 \text{ และ } n = 4 \text{ ดังนั้น } \exp(2^4) = 2 * 2 * 2 * 2 = 16$$

ตัวอย่างโปรแกรมที่ 7.19 แสดงการสร้างและใช้งานฟังก์ชันเวียนเกิด

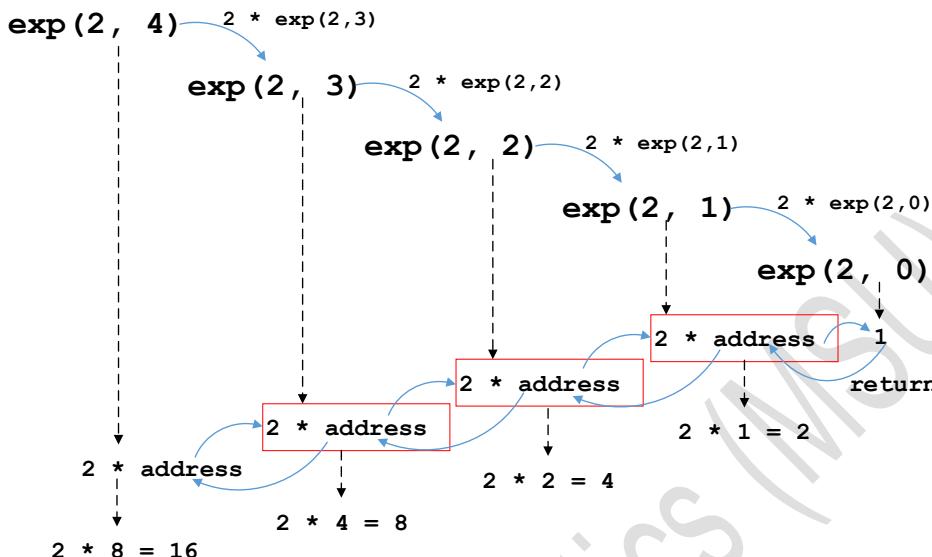
Program Example 7.19: Recursion

```
1 # Computing exponent
2 def exp(x, n):
3     if n == 0:
4         return 1      #constant (minimum solution)
5     else:
6         return x * exp(x, n-1)
7 print("Exponent of 2^4 = ", exp(2, 4))
```



Exponent of $2^4 = 16$

อธิบายการทำงานของฟังก์ชันเรียกตัวเอง exp ดังรูปที่ 7.5



รูปที่ 7.5 แสดงการทำงานของฟังก์ชันเรียกตัวเอง (exp)

จากรูปที่ 7.5 เมื่อโปรแกรมหลักทำการเรียกฟังก์ชัน `exp` พร้อมกับส่งค่าคงที่เป็นอาร์กิวเม้นต์เข้าไปในฟังก์ชัน 2 ค่า คือ เลขฐาน = 2 และเลขยกกำลัง = 4 (ฐานมีค่าเท่ากับ 2 และเลขยกกำลังมีค่าเท่ากับ 4) ฟังก์ชัน `exp` จะเริ่มทำการคำนวนโดยเริ่มต้นจากการนำเอา $2 * \exp(2, 3)$ ซึ่งเป็นการเรียกตัวเองซ้ำรอบที่ 1 พร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์ 2 ตัวคือ เลขฐาน = 2 และเลขยกกำลัง = 3 ผลที่ได้คือ $2 * (2 * \exp(2, 2))$ จากนั้นโปรแกรมทำการเรียกซ้ำตัวเองเป็นรอบที่ 2 พร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์ 2 ตัวคือ เลขฐาน = 2 และเลขยกกำลัง = 2 ผลที่ได้คือ $2 * (2 * (2 * \exp(2, 1)))$ จากนั้นโปรแกรมทำการเรียกซ้ำตัวเองเป็นรอบที่ 3 พร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์ 2 ตัวคือ เลขฐาน = 2 และเลขยกกำลัง = 1 ผลที่ได้คือ $2 * (2 * (2 * (2 * \exp(2, 0))))$ ในรอบนี้ เงื่อนไขในคำสั่ง `if` เป็นจริง เพราะว่าค่า `n` มีค่าเท่ากับ 0 ($2^0 = 1$) ส่งผลให้มีการส่งค่ากลับ เป็นค่าคงที่เท่ากับ 1 ออกมา (ค่าดังกล่าวเป็นค่าตอบของปัญหาที่เล็กที่สุด) จากการส่งค่ากลับเป็น 1 ออกมา ทำให้โปรแกรมเริ่มประมวลผลย้อนกลับอัตโนมัติ ดังนี้

$$2 * (2 * (2 * (2 * \exp(2, 0))))$$

เมื่อ $\exp(2, 0) = 1$ ดังนั้น \Rightarrow

$$2 * (2 * (2 * (2 * 1)))$$

$$(2 * 1) = 2 \Rightarrow$$

$$2 * (2 * (2 * 2))$$

$$(2 * 2) = 4 \Rightarrow$$

$$2 * (2 * 4)$$

$$(2 * 4) = 8 \Rightarrow 2 * 8$$

$$(2 * 8) \Rightarrow 16$$

ตัวอย่างโจทย์ปัญหาที่คลาสสิกอีก 2 ตัวอย่าง คือ Factorial และ Fibonacci ซึ่งมีสมการดังนี้

$$\text{Factorial } (n) = n! = \begin{cases} 1, & x = 0 \\ \prod_{k=1}^n k, & x > 0 \end{cases}$$

$$\text{เช่น } 5! = 5 * 4 * 3 * 2 * 1 = 120$$

$$\text{Fibonacci} = F_n = F_{n-1} + F_{n-2} \text{ โดยที่ } F_0 = 0 \text{ และ } F_1 = 1$$

$$\text{เช่น } F_5 = F_4 + F_3, F_4 = F_3 + F_2, F_3 = F_2 + F_1, F_2 = F_1 + F_0$$

เมื่อ $F_0 = 0$ และ $F_1 = 1$ ดังนั้น

$$F_2 = 1 + 0 = 1, F_3 = 1 + 1 = 2, F_4 = 2 + 1 = 3, F_5 = 3 + 2 = 5$$

$$\text{ดังนั้น } F_5 = 5$$

ตัวอย่างโปรแกรมของ Factorial และ Fibonacci และใน โปรแกรมที่ 7.20

ตัวอย่างโปรแกรมที่ 7.20 Factorial และ Fibonacci

Program Example 7.20: Factorial Vs Fibonacci

```

1 # Factorial & Fibonacci
2 #Factorial function
3 def factorial(n):
4     if n == 1:
5         return 1      #constant (minimum solution)
6     else:
7         res = n * factorial(n-1)
8         return res
9
10 #Fibonacci function
11 def fib(n):
12     if n == 0:
13         return 0      #constant (minimum solution)
14     elif n == 1:
15         return 1      #constant (minimum solution)
16     else:
17         return fib(n-1) + fib(n-2)
18 #call both functions
19 print("Factorial (5) = ",factorial(5))
20 print("Fibonacci (5) = ",fib(5))

```



OUTPUT

```
Factorial (5) = 120  
Fibonacci (5) = 5
```

จบบทที่ 7

Faculty of Informatics (MSU)

บทที่ 8

โมดูล และแพ็คเกจ

(Modules and Packages)



1. โมดูลคืออะไร?

โมดูล คือ การจัดกลุ่มของแฟ้มต้นฉบับที่ถูกเขียนด้วยภาษาไพธอนอย่างเป็นระบบตามภาระหน้าที่ที่ถูกกำหนดไว้ และสามารถเรียกใช้งานได้โดยสะดวก เช่น โมดูลที่บริหารจัดการเกี่ยวกับระบบปฏิบัติการ (Operating system: os) โมดูลที่ใช้สำหรับประมวลผลทางคณิตศาสตร์ (Mathematics: math) เป็นต้น โดยกลุ่มของแฟ้มที่ถูกเก็บรวมรวมไว้อาจจะประกอบไปด้วย พังชัน คลาส ตัวแปร หรือ runnable code ก็ได้ โมดูลมี 2 ประเภทคือ โมดูลที่มาพร้อมกับตัวภาษาไพธอน เช่น โมดูล io, math, os, random, re, socket, string, sys, types, xml เป็นต้น และโมดูลที่ผู้เขียนโปรแกรมเขียนขึ้นมาใช้งานเอง โดยปกติชื่อโมดูลจะมีชื่อเดียวกับชื่อแฟ้มที่จัดเก็บโมดูลไว้ แต่ถ้าแฟ้มนั้นมีหลาย ๆ โมดูล หรือหลายพังชัน ผู้เขียนโปรแกรมควรตั้งชื่อแฟ้มให้สอดคล้องกับงานที่ทำ ตัวอย่างเช่น โมดูล CalAreaRectangle ทำหน้าที่คำนวณหาพื้นที่รูปสี่เหลี่ยมใดๆ ประกอบด้วยพังชันคำนวณพื้นที่สี่เหลี่ยมด้านขนาน (Parallelogram) ผืนผ้า (Rectangle) ด้านเท่า (Square) คางหมู (Trapezoid) เปียกปูน (Rhombooid) รูปว่าว (KiteRectangular) และสี่เหลี่ยมใดๆ (AnyRectangular) เมื่อสร้างพังชันแล้วให้ทำการบันทึกแฟ้มเป็นชื่อ CalAreaRectangle.py

ตัวอย่างโปรแกรมที่ 8.1 โมดูล CalAreaRectangle

Program Example 8.1: Module

```
1 # Calculating area of any rectangles
2 # This module named CalAreaRectangle.py
3 def rectangle(width, height):
4     return width * height
5
6 def square(width1, width2):
7     return width1 * width2
8
9 def parallelogram(height, base):
10    return height * base
11
12 def trapezoid(sumofpararell, height):
```

```

13     return 0.5 * sumofpararell * height
14
15 def rhomboid(mulofdiagonal):
16     return 0.5 * mulofdiagonal
17
18 def RectangularKite(mulofdiagonal):
19     return 0.5 * mulofdiagonal
20
21 def AnyRectangular(diagonal, sumofbranch):
22     return 0.5 * diagonal * sumofbranch

```

2. การเรียกใช้งานโมดูล

เมื่อผู้เขียนโปรแกรมต้องการเรียกใช้ฟังก์ชันที่รวมเป็นโมดูลไว้แล้ว สามารถเรียกโดยใช้คำสั่ง import และนิยมเขียนคำสั่งดังกล่าวไว้ตั้งแต่บรรทัดแรกของโปรแกรม สำหรับวิธีการเรียกใช้ฟังก์ชันในโมดูลสามารถกระทำได้ 4 วิธี ได้แก่ ใช้คำสั่ง import, from module import function, import module import * และ import OldModule as NewModule ซึ่งมีรูปแบบการเรียกใช้ดังต่อไปนี้

- การใช้คำสั่ง import

```
import module1[, module2[, ... moduleN]]
```

โดย import คือ คำสั่งเรียกใช้โมดูล, module1 คือ ชื่อโมดูลที่ต้องการเรียกใช้งาน และ module2...moduleN คือ ชื่อโมดูลอื่นๆ ที่ต้องการเรียกใช้เพิ่มเติม (กรณีใช้งานมากกว่า 1 โมดูล)

หลักการใช้งาน: เหมาะสำหรับผู้ที่ต้องการเรียกใช้งานฟังก์ชันและตัวแปรทั้งหมดในโมดูลเข้ามาทำงานในโปรแกรม

ตัวอย่าง เช่น

```
import sys, io, math
```

- การใช้คำสั่ง from module import function

```
from moduleName import funcName1[, FuncName2[, ... FuncNameN]]
```

โดย from moduleName import คือ คำสั่งเรียกใช้โมดูลชื่อ moduleName, funcName1,...,N คือ ชื่อของฟังก์ชันที่อยู่ในโมดูล moduleName

หลักการใช้งาน: เหมาะสำหรับผู้ที่ต้องการเรียกฟังก์ชันหรือตัวแปรเฉพาะที่ต้องการมาใช้งานเท่านั้น

ตัวอย่าง

```
from math import pi, pow
```

- การใช้คำสั่ง from module import *

```
from moduleName import *
```

โดย `from moduleName import` คือ คำสั่งเรียกใช้โมดูลชื่อ `moduleName`, `*` คือ พังชันทั้งหมดที่อยู่ในโมดูล `moduleName`

หลักการใช้งาน: เมมาะสำหรับผู้ที่ต้องการเรียกพังชันหรือตัวแปรทั้งหมดในโมดูลมาทำงาน

ตัวอย่าง

```
from math import *
```

- การใช้คำสั่ง `import OldModule as NewModule`

```
import OldModule as NewModule
```

โดย `import OldModule` คือ คำสั่งเรียกใช้โมดูลชื่อ `OldModule` และ `as NewModule` คือ โมดูลใหม่ที่ถูกเปลี่ยนชื่อจาก `OldModule` ไปเป็น `NewModule`

หลักการใช้งาน: เมมาะสำหรับผู้ที่ต้องการเปลี่ยนชื่อโมดูลเดิม ให้เป็นชื่อใหม่ ตัวอย่าง

```
import math as MAT
```

`print(MAT.pi)` เปลี่ยนชื่อโมดูลจาก `math` เป็นชื่อ `MAT`

หรือใช้คำสั่ง `from module import OldName as NewName` เช่น

```
from math import sqrt as SquareRoot
```

เปลี่ยนชื่อพังชันจาก `sqrt` เป็น `SquareRoot`



Caution! เมื่อฯพรอนแปลคำสั่งมาถึงคำสั่ง `import` และฯพรอนจะทำการโหลดโมดูลที่ระบุไว้ขึ้นมาทำงาน โดยค้นหาที่อยู่ของโมดูลจาก `path` ซึ่งถูกกำหนดไว้ในระบบปฏิบัติการ ถ้าไฟรอนค้นหาแฟ้มดังกล่าวไม่พบ จะทำให้โปรแกรมเกิดข้อผิดพลาด (สำหรับวินโดว์กำหนด `path` ที่ `system` \Rightarrow `Advanced system settings` \Rightarrow `Environment variables` \Rightarrow `System variables` \Rightarrow `Path`)

โปรแกรมที่ 8.2 แสดงตัวอย่างการเรียกใช้โมดูล `CalAreaRectangle` โดยใช้คำสั่ง `import` (ผู้เขียนโปรแกรมควรเก็บแฟ้ม `CalAreaRectangle.py` ไว้ในไดเรกทอรีเดียวกับโปรแกรมที่เรียกใช้งาน)

ตัวอย่างโปรแกรมที่ 8.2 การเรียกใช้โมดูลด้วยคำสั่ง `import`

Program Example 8.2: calling CalAreaRectangle module	
1	<code># import CalAreaRectangle module</code>
⇒2	<code>import CalAreaRectangle</code>
⇒3	<code>print("Area of Rectangle : ",CalAreaRectangle.rectangle(3, 4))</code>
4	<code>print("Area of Squre : ",CalAreaRectangle.squre(3, 3))</code>
5	<code>print("Area of Parallelogram : ",CalAreaRectangle.parallelogram(1.5, 4.5))</code>
6	<code>print("Area of Trapezoid : ",CalAreaRectangle.trapezoid(5, 2.5))</code>
7	<code>print("Area of Rhomboid : ",CalAreaRectangle.rhomboide(8))</code>

```

8   print("Area of RectangularKite :",CalAreaRectangle.
9     RectangularKite(12))
print("Area of AnyRectangular
: ",CalAreaRectangle.AnyRectangular(3.5, 6)

```



OUTPUT

```

Area of Rectangle : 12
Area of Squre : 9
Area of Parallelogram : 6.75
Area of Trapezoid : 6.25
Area of Rhomboid : 4.0
Area of KiteRectangular : 6.0
Area of AnyRectangular : 10.5

```

จากตัวอย่างโปรแกรมที่ 8.2 บรรทัดที่ 2 แสดงตัวอย่างการใช้คำสั่ง import โมดูล

CalAreaRectangle เข้ามาทำงาน เมื่อต้องการเรียกใช้ฟังชันใดๆ ในโมดูลดังกล่าว ผู้ใช้จำเป็นต้องอ้างชื่อโมดูล แล้วตามด้วยชื่อฟังชันที่ต้องการใช้งาน (โดยใช้ . ในการอ้างอิงถึงฟังชัน) เช่น ในบรรทัดที่ 3 ต้องการเรียกใช้ฟังชัน CalAreaRectangle.rectangle พร้อมกับส่งค่าคงที่เป็นอาร์กิวเม้นต์ให้กับฟังชัน 2 ตัวคือ 3 และ 4 สำหรับการเรียกใช้ฟังชันอื่นๆ ก็ทำในลักษณะเดียวกัน สำหรับตัวอย่างที่ 8.3 แสดงการใช้คำสั่ง from module import function

ตัวอย่างโปรแกรมที่ 8.3 การเรียกใช้โมดูลด้วยคำสั่ง from module import function

Program Example 8.2: calling CalAreaRectangle module

```

1  # from moduleName import funcName example
2  from CalAreaRectangle import rectangle, squre, parallelogram,
3    trapezoid, rhomboid, RectangularKite, AnyRectangular
4  print("Area of Rectangle :",rectangle(3, 4))
5  print("Area of Squre :",squre(3, 3))
6  print("Area of Parallelogram :",parallelogram(1.5, 4.5))
7  print("Area of Trapezoid :",trapezoid(5, 2.5))
8  print("Area of Rhomboid :",rhomboid(8))
9  print("Area of KiteRectangular :", RectangularKite(12))
print("Area of AnyRectangular :",AnyRectangular(3.5, 6))

```



OUTPUT

```

Area of Rectangle : 12
Area of Squre : 9
Area of Parallelogram : 6.75
Area of Trapezoid : 6.25
Area of Rhomboid : 4.0
Area of KiteRectangular : 6.0
Area of AnyRectangular : 10.5

```

จากตัวอย่างโปรแกรมที่ 8.3 บรรทัดที่ 2 แสดงการใช้คำสั่ง from module import function

โปรแกรมเริ่มต้นโดยการ import โมดูล CalAreaRectangle เข้ามา ก่อนโดยใช้คำสั่ง from CalAreaRectangle จากนั้นทำการเลือกเอาเฉพาะฟังชันที่ต้องการใช้งานจริงๆ มาทำงานเท่านั้น โดยใช้คำสั่ง import rectangle, squre, parallelogram, trapezoid, rhomboid, RectangularKite,

AnyRectangular เมื่อโปรแกรมต้องการเรียกใช้ฟังชัน ไม่จำเป็นต้องอ้างชื่อโมดูลเหมือนการใช้คำสั่ง
`import` แบบปกติ เช่น `print("Area of Rectangle :", rectangle(3, 4))` ในบรรทัดที่ 3 เป็นต้น



Tips: เมื่อต้องการเรียกใช้ฟังชันในโมดูลโดยไม่ต้องการอ้างชื่อของโมดูล ให้ใช้คำสั่ง `form ModuleName import function` และสามารถเลือกเอาเฉพาะฟังชันที่ต้องการใช้จริงๆ เท่านั้น
 เมื่อผู้เขียนโปรแกรมต้องการดูรายละเอียดของฟังชัน ตัวแปร ภายในโมดูลที่โหลดเข้ามาใช้งานใน
 โปรแกรมสามารถใช้คำสั่ง `dir(ModuleName)` ดังตัวอย่างต่อไปนี้

```
>>> import CalAreaRectangle
>>> dir(CalAreaRectangle)
['AnyRectangular', 'KiteRectangular', '__builtins__',
 '__cached__', '__doc__', '__file__', '__initializing__',
 '__loader__', '__name__', '__package__', 'parallelogram',
 'rectangle', 'rhomboid', 'square', 'trapezoid']
```

โดยตัวแปร `__name__` เก็บชื่อของโมดูล (แสดงชื่อโดยใช้คำสั่ง `print(CalAreaRectangle.__file__)`) และตัวแปร `__file__` เก็บชื่อของแฟ้มที่โหลดมาใช้งาน (แสดงชื่อโดยใช้คำสั่ง `print(CalAreaRectangle.__file__)`) มีนามสกุลเป็น .py



Tips: เมื่อผู้เขียนโปรแกรม `import` โมดูลใดโมดูลหนึ่งเข้ามาในโปรแกรมแล้ว สามารถ
 เรียกดูฟังชันในโมดูลเหล่านั้นด้วยคำสั่ง `dir(ModuleName)` เช่น `dir(CalAreaRectangle)`
 คำสั่ง `dir()` สามารถเรียกดูรายละเอียดของโมดูลที่ติดตั้งมากับ Python ได้เช่นเดียวกัน

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', 'acos',
 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc',
 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf', 'isnan',
 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf',
 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
 'trunc']
```

ความแตกต่างระหว่าง `import`, `from module import function` และ `import *`

สำหรับความแตกต่างระหว่างคำสั่ง `import` และ `from module import function` คือ `import` จะ¹ โหลดโมดูล พร้อมกับฟังชันทั้งหมดเข้ามาในโปรแกรมที่เรียกใช้งาน แต่ `from module import function` จะสามารถเลือกเอาเฉพาะฟังชันที่จำเป็นต้องใช้งานเข้ามาเท่านั้น ส่งผลให้ประหยัดพื้นที่หน่วยความจำในการทำงานลงได้ แต่ผู้เขียนโปรแกรมต้องจดจำให้ได้ว่าแต่ละโมดูลมีฟังชันให้เรียกใช้งานอะไรบ้าง

สำหรับคำสั่ง from module import * จะใช้มือผู้เขียนโปรแกรมต้องการนำเข้าฟังชันทั้งหมดในโมดูลเข้ามาทำงานในโปรแกรม ซึ่งไม่แตกต่างกับคำสั่ง import moduleName เพราะฉนั้น Python แนะนำให้ใช้คำสั่งนี้เท่าที่จำเป็นเท่านั้น สำหรับการใช้งาน import OldModule as NewModule และ from module import OldName as NewName เป็นการเปลี่ยนชื่อโมดูล และฟังชันจากชื่อเดิมเป็นชื่อใหม่เท่านั้น

ตำแหน่งที่อยู่ของการโหลดโมดูลมาใช้งาน

เมื่อผู้เขียนโปรแกรมโหลดโมดูลใดโมดูลหนึ่งเข้ามาทำงานในโปรแกรม ตัวแปลภาษา Python จะเริ่มทำการค้นหาโมดูลตามลำดับดังนี้

1. ค้นหาแฟ้มที่เก็บโมดูลในไดเรคทรอรีปัจจุบัน เมื่อไม่พบโมดูลที่ต้องการจะค้นหาต่อในลำดับที่ 2
2. Python จะค้นหาแฟ้มในแต่ละไดเรคทรอรีที่ถูกระบุใน PYTHONPATH ถ้าไม่พบโมดูลที่ต้องการจะค้นหาต่อไปในลำดับที่ 3
3. Python จะค้นหาใน default path ซึ่งถูกกำหนดโดยระบบปฏิบัติการที่ติดตั้ง Python ไว้ เช่น ถ้าติดตั้งไว้บนวินโดว์ส default path จะถูกกำหนดไว้ที่ C:\PythonXX (โดย XX คือเวอร์ชันของ Python เช่น Python34) แต่ถ้าเป็น UNIX โดยปกติจะอยู่ที่ /usr/local และ /lib/python ถ้า Python ยังหาโมดูลไม่เจอใน path ที่ระบุไว้ทั้งหมด จะเกิด error ขึ้น และไม่สามารถใช้งานโมดูลนั้นๆ ได้

เมื่อผู้เขียนโปรแกรมต้องการดู default path ที่ Python ได้กำหนดไว้ สามารถใช้คำสั่งดังนี้

```
>>> import sys  
>>> print(sys.path)  
['C:\\\\Python34\\\\Lib\\\\idlelib',  
 'C:\\\\Windows\\\\SYSTEM32\\\\python34.zip', 'C:\\\\Python34\\\\DLLs',  
 'C:\\\\Python34\\\\lib', 'C:\\\\Python34', 'C:\\\\Python34\\\\lib\\\\site-  
 packages']
```

ผู้เขียนโปรแกรมสามารถเพิ่มเติมที่อยู่ของโปรแกรมหรือโมดูลที่เขียนขึ้นได้เอง โดยใช้คำสั่ง sys.path.append(PATH_NAME) เช่น

```
>>> import sys  
>>> sys.path.append('C:\\\\test')  
>>> print(sys.path)  
['C:\\\\Python33\\\\Lib\\\\idlelib',  
 'C:\\\\Windows\\\\SYSTEM32\\\\python33.zip', 'C:\\\\Python33\\\\DLLs',  
 'C:\\\\Python33\\\\lib', 'C:\\\\Python33', 'C:\\\\Python33\\\\lib\\\\site-  
 packages', 'C:\\\\test']
```

PYTHONPATH

ตัวแปร PYTHONPATH เป็นตัวแปรระบบ ที่ถูกสร้างขึ้นจากการระบุพื้นที่การ ซึ่งตัวแปร ดังกล่าวจะเก็บบัญชีรายชื่อที่อยู่ของ ไดเรกทอรีที่ระบบใช้สำหรับทำงาน รวมถึงไดเรกทอรีของไฟร่อน ด้วย ตัวแปรดังกล่าวสามารถกำหนดใหม่ได้ โดยใช้รูปแบบคำสั่งคือ

บนระบบปฏิบัติการวินโดว์ เลือก start ⇒ run ⇒ cmd

```
set PYTHONPATH=c:\python34\lib;
```

บนระบบปฏิบัติการยูนิกซ์ เปิด terminal

```
set PYTHONPATH=/usr/local/lib/python
```

บนระบบปฏิบัติการวินโดว์ สามารถกำหนดผ่านกราฟฟิกได้ โดยเลือก My Computer ⇒ คลิกขวาเลือก properties ⇒ เลือก System ⇒ Advanced system settings ⇒ Environment variables.. ⇒ System variables ⇒ Path ⇒ edit ⇒ ให้กำหนด path ดังนี้คือ C:\YourPath เช่น %C:\Python34 ⇒ กดปุ่ม OK ไปเรื่อยๆ จนกว่าหน้าต่างการทำงานจะหมด

การโหลดโมดูลมาใช้งานใหม่

โดยปกติเมื่อผู้เขียนโปรแกรมทำการนำเข้าโมดูลใดๆ เข้ามาทำงานในโปรแกรม นิยมประกาศไว้ บรรทัดบนสุดของโปรแกรม โดยโมดูลนั้นๆ จะถูกโหลดมาใช้งานเพียงครั้งเดียวเท่านั้น (ไฟรอนแนะนำให้โหลดโมดูลเพียงแค่ครั้งเดียวจนกว่าโปรแกรมจะหยุดการทำงาน) จนโปรแกรมยุติการทำงาน แต่ถ้า ผู้เขียนโปรแกรมต้องการโหลดโมดูลได้โมดูลหนึ่งให้ทำงานอีกครั้งในโปรแกรม มีรูปแบบคำสั่งดังนี้

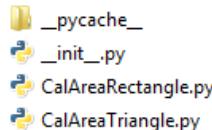
```
reload(module_name)
```

ตัวอย่างเช่น reload(CalAreaRectangle)

3. แพ็คเกจ (Packages)

โปรแกรมที่มีขนาดใหญ่ จะประกอบด้วยคลาส โมดูล และฟังชันอยู่เป็นจำนวนมาก เราสามารถ จัดหมวดหมู่ ของคลาส โมดูล และฟังชันให้เป็นระเบียบและง่ายต่อการใช้งาน โดยการนำเอคลาส และ โมดูลเหล่านั้น รวมเข้ากันไว้ตามลำดับชั้นด้วยโครงสร้างแฟ้มอย่างเป็นระบบ และใช้คำสั่ง import ใน การเรียกใช้ สำหรับการสร้างแพ็คเกจมีขั้นตอนดังนี้

จากที่กล่าวมาแล้วข้างต้น ผู้เขียนได้สร้างโมดูลและเก็บไว้ในแฟ้มชื่อว่า CalAreaRectangle.py ไว้แล้ว ต่อไปผู้เขียนจะสร้างโมดูลเพิ่มขึ้นอีก 1 โมดูลคือ โมดูลคำนวณหาพื้นที่สามเหลี่ยม (CalAreaTriangle) และดูในโปรแกรมที่ 8.4 ซึ่งประกอบไปด้วย สามเหลี่ยมทั่วไป (Triangle) สามเหลี่ยมด้านเท่า (Equilateral) สามเหลี่ยมหน้าจั่ว (Isosceles) และสามเหลี่ยมนูนจาก (Pythagorean) และทำการบันทึกเป็นแฟ้มชื่อ CalAreaTriangle.py และเก็บโมดูลทั้งสอง (CalAreaRectangle และ CalAreaTriangle) ไว้ในไดเรกทรอรีชื่อ CalArea ดังรูปด้านล่าง



ตัวอย่างโปรแกรมที่ 8.4 โมดูล CalAreaTriangle

Program Example 8.4: CalAreaTriangle module

```

1 # Calculating Triangles Area
2 # This module named CalAreaTriangle.py
3 def triangle(height, base):
4     return 1/2 * height * base
5
6 def equilateral(width):
7     return 3** (1/2) * width * width
8
9 def isosceles(base, sidebase):
10    return base/4 * 4 * (((sidebase*sidebase) -
11 (base*base)) **(1/2))
12
13 def pythagorean(perpendicular):
14    return 0.5 * perpendicular * perpendicular

```

ขั้นตอนต่อไป ให้ผู้เขียนโปรแกรมสร้างแฟ้มชื่อ `__init__.py` เพื่อใช้สำหรับเก็บค่าคงพิกัดต้นสำหรับเรียกใช้งานแพ็คเกจ CalArea และเก็บไว้ในไดเรกทรอรีชื่อ CalArea ร่วมกันกับแฟ้ม CalAreaRectangle.py และ CalAreaTriangle.py จากนั้นทำการกำหนดค่าให้กับแฟ้ม `__init__.py` ดังนี้

```

#This initial file for CalArea Package (__init__.py)
from .CalAreaRectangle import rectangle, square, parallelogram,
trapezoid, rhomboid, KiteRectangular, AnyRectangular

from .CalAreaTriangle import triangle, equilateral, isosceles,
pythagorean

```

ในแฟ้ม `__init__.py` ให้ผู้เขียนโปรแกรมทำการ `import` โมดูลพร้อมกับพังชันทั้งหมดที่ต้องการใช้งาน ดังตัวอย่างที่แสดงไว้ข้างบน จากนั้นทดสอบเขียนโปรแกรมเพื่อเรียกใช้งานแพ็คเกจ CalArea โดยใช้คำสั่ง `import` และตามด้วยชื่อของแพ็คเกจ (สำหรับในตัวอย่างคือ `import CalArea`) ดังตัวอย่าง โปรแกรมที่ 8.5

ตัวอย่างโปรแกรมที่ 8.5 import CalArea

Program Example 8.5: calling CalArea Area

```

1   # Calling package CalArea
⇒2  import CalArea    # import package CalArea
⇒3  print("Area of square :",CalArea.square(2.5, 2.5))
⇒4  print("Area of rectangle :",CalArea.rectangle(2.5, 3.5))
⇒5  print("Area of triangle :",CalArea.triangle(3, 2.5))
⇒6  print("Area of equilateral :",CalArea.equilateral(2.5))

```



Area of square : 6.25
 Area of rectangle : 8.75
 Area of triangle : 3.75
 Area of equilateral :
 10.825317547305481

จากโปรแกรมตัวอย่างที่ 8.5 บรรทัดที่ 2 โปรแกรมทำการ import แพ็กเกจชื่อ CalArea เข้ามาทำงาน จากนั้นในบรรทัดที่ 3, 4, 5 และ 6 ทดสอบเรียกฟังชันที่อยู่ภายใต้แพ็กเกจดังกล่าว โดยผู้ใช้ต้องทราบด้วยว่าแต่ละฟังชันต้องการอาร์กิวเม้นต์ในการทำงานกี่ตัว อะไรบ้าง ผลลัพธ์จากการเรียกใช้ฟังชันแสดงในตัวอย่างข้างบน



Caution! ระวัง ในไดเรคทรอรีที่สร้างเป็นแพ็กเกจ (Package) จะต้องประกอบด้วยแฟ้ม (.py) ที่มีนามสกุล *.py และ __init__.py โดยต้องสะกดชื่อแฟ้ม __init__ ให้ถูกต้องเสมอ สรุปการสร้างและใช้งานแพ็กเกจ

- สร้างโมดูลที่ประกอบไปด้วยฟังชันต่างๆ ที่ผู้เขียนโปรแกรมต้องการ เช่น โมดูล CalAreaRectagle ประกอบด้วยฟังชันคำนวณพื้นที่สี่เหลี่ยมด้านเท่า (Square) ผืนผ้า (Rectangle) เป็นต้น เสร็จแล้วบันทึกแฟ้มเป็นชื่อ CalAreaRectagle.py
- สร้างแฟ้มเพื่อกำหนดค่าคอนฟิกให้กับแพ็กเกจ ชื่อ __init__.py (ต้องใช้ชื่อนี้เท่านั้นและต้องสะกดชื่อแฟ้มให้ถูกต้องด้วย) ภายในแฟ้มให้ทำการ import โมดูลและฟังชันที่ต้องการใช้งาน เช่น from .CalAreaRectagle import square, rectangle เป็นต้น
- สร้างไดเรคทรอรีเพื่อทำเป็นแพ็กเกจสำหรับโมดูลที่สร้างขึ้น โดยตั้งชื่อไดเรคทรอรีให้สอดคล้องกับโมดูลที่สร้าง ในตัวอย่างนี้จะสร้างไดเรคทรอรีชื่อ CalArea จากนั้นให้ทำการคัดลอกแฟ้ม CalAreaRectagle.py และ __init__.py เข้าไปในไดเรคทรอรีที่สร้างขึ้น (ถ้ามีมากกว่า 1 แฟ้มให้คัดลอกเข้าไปทั้งหมด)
- เขียนโปรแกรมทดสอบใช้งานแพ็กเกจ โดยใช้คำสั่ง import เข้ามารаботา เช่น

```

import CalArea      # import package CalArea
print("Area of square :", CalArea.square(2.5, 2.5)) # calling the method in module

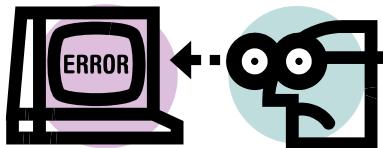
```

จบบทที่ 8

บทที่ 9

การจัดการข้อผิดพลาด

(Handling Exceptions)



1. ข้อผิดพลาด (Exceptions) คืออะไร?

ข้อผิดพลาด คือ เหตุการณ์ที่เกิดขึ้นขณะโปรแกรมกำลังประมวลผลและส่งผลให้เกิดการขัดจังหวะ หรือรบกวนการประมวลผลคำสั่งปักดิ่งของโปรแกรม โดยทั่วไปแล้วความผิดพลาดสามารถแบ่งออกเป็น 2 ประเภท คือ ความผิดพลาดขณะคอมไพล์ (Compile-Time Errors) หรือ Syntax Error และความผิดพลาดขณะโปรแกรมกำลังทำการประมวลผล (Run- Time-Errors)

- 1) ความผิดพลาดขณะคอมไпал์ เกิดขึ้นจากผู้เขียนโปรแกรมไม่ปฏิบัติตามข้อกำหนดตามไวยกรณ์ของภาษา เช่น สะกดประโยคคำสั่งผิดพลาด ประกาศตัวแปรที่เหมือนกับคำสงวน เป็นต้น ลักษณะความผิดพลาดแบบนี้ผู้เขียนโปรแกรมจะพบเมื่อทำการสั่งคอมไพล์โปรแกรม ถ้าไม่มีการแก้ไขข้อผิดพลาดเหล่านี้ก่อน โปรแกรมจะไม่สามารถทำงานได้
- 2) ความผิดพลาดขณะทำการประมวลผล สามารถจำแนกได้เป็น 2 สาเหตุใหญ่ๆ คือ ความผิดพลาดจากวิธีการคิด (Program Logic) และความผิดพลาดที่เกิดจากสภาวะแวดล้อมของการทำงานไม่สมบูรณ์
 - ความผิดพลาดจากวิธีการคิด คือ ผู้เขียนโปรแกรมเกิดความเข้าใจผิดเกี่ยวกับกระบวนการทำงานของโปรแกรม เช่น โปรแกรมทำการหารด้วยศูนย์ โปรแกรมสั่งอ่านแฟ้มแต่ไม่ปรากฏแฟ้มดังกล่าวในระบบ เป็นต้น (ในส่วนนี้ผู้เขียนโปรแกรมสามารถบริหารจัดการได้ โดยใช้คำสั่ง try...except)
 - ความผิดพลาดที่เกิดจากสภาวะแวดล้อมการทำงานไม่สมบูรณ์ เช่น หน่วยความจำไม่เพียงพอต่อการทำงาน ระบบเครือข่ายไม่พร้อมใช้งาน พื้นที่ที่เก็บข้อมูลเต็ม เป็นต้น

โดยปกติเมื่อโปรแกรมที่เขียนขึ้นเกิดข้อผิดพลาดอย่างใดอย่างหนึ่ง ไฟรอนจะจัดการความผิดพลาดเหล่านั้นทันที เรียกว่ากระบวนการนี้ว่า การจัดการข้อผิดพลาด (Exceptions handling) แต่ถ้าไม่สามารถจัดการความผิดพลาดเหล่านั้นได้ ไฟรอนจะหยุดการทำงานของโปรแกรมเหล่านั้นทันที สำหรับการบริหารจัดการความผิดปกติที่ไม่ได้คาดการณ์ไว้ ไฟรอนได้จัดเตรียมคำสั่งไว้ให้ 2 วิธีคือ การจัดการข้อผิดพลาด (Exceptions handling) และการยืนยันในสมมติฐาน (Assertions)

2. การจัดการข้อผิดพลาด (Exceptions handling)

คุณลักษณะที่สำคัญอย่างมากสำหรับการเขียนโปรแกรมคือ ความคงทนของโปรแกรม โปรแกรมที่มีความคงทนจะไม่ล้มเหลวหรือหยุดการทำงาน แม้ในขณะที่เกิดข้อผิดพลาดขึ้นจากความผิดพลาดของผู้ใช้ หรือจากสิ่งแวดล้อมอื่นๆ เพื่อให้โปรแกรมสามารถต่อสู้กับความล้มเหลว ในขณะทำงานได้ ไฟรอนได้จัดเตรียมคำสั่งในการจัดการความผิดพลาดไว้ โดยมีรูปแบบคำสั่งดังนี้

```

try:
    normal statement(s)
except Exception 1:
    exception statement(s)
except Exception 2:
    exception statement(s)
    .....
except Exception n:
    exception statement(s)
else:
    normal statement(s)

```

เมื่อนำมาใช้คำสั่ง try...except มีดังนี้

- โปรแกรมตรวจส่วนที่ผู้เขียนโปรแกรมคาดว่าจะเกิดข้อผิดพลาดขึ้นได้ในอนาคต (normal statement(s)) ให้ใช้คำสั่ง try ครอบไว้
- คำสั่ง try จะทำงานร่วมกับ except เสมอ โดยคำสั่ง except สามารถใช้งานได้มากกว่า 1 ครั้ง เนื่องจากการตรวจสอบความผิดพลาดที่เกิดขึ้น มีได้หลายกรณี
- ผู้เขียนโปรแกรมสามารถแสดงข้อผิดพลาดของโปรแกรมได้เอง โดยกำหนดไว้หลังคำสั่ง except (exception statement(s)) เช่น `except IOError: print ("Error: can't find file or read data")`
- ไฟรอนออกแบบให้ผู้เขียนโปรแกรมสามารถใช้คำสั่ง else ปิดท้ายคำสั่ง try...except ได้ สำหรับคำสั่ง else จะทำงานก็ต่อเมื่อโปรแกรมไม่มีข้อผิดพลาดใดๆ เกิดขึ้น

ตัวอย่างโปรแกรมที่ 9.1 แสดงตัวอย่างการทำงานของ try...except

Program Example 9.1: try...except	
--	--

1	# Try...except first program
⇒2	try:

```

⇒3   fh = open("myfile", "w")
⇒4       fh.write("This is my file for exception handling!!")
⇒5   except IOError:
6       print ("Error: can't find file or read data")
⇒7   else:
8       print ("Written content in the file successfully")
⇒9   fh.close()

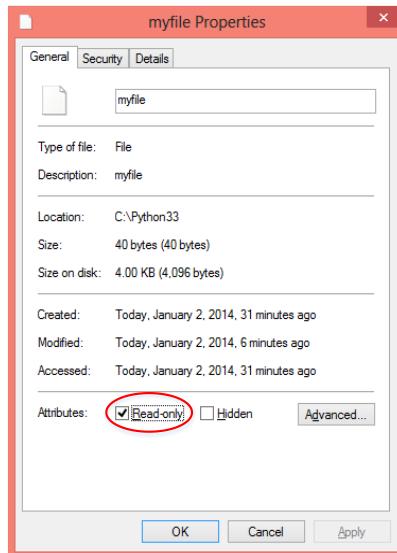
```



Written content in the file successfully

จากโปรแกรมตัวอย่างที่ 9.1 แสดงตัวอย่างการใช้งาน try...except บรรทัดที่ 2 โปรแกรมประกาศคำสั่ง try: เพื่อทำหน้าที่ตรวจสอบความผิดพลาดที่อาจจะเกิดขึ้นกับการเปิดแฟ้มข้อมูลในบรรทัดที่ 3 และสั่งเขียนแฟ้มในบรรทัดที่ 4 จากตัวอย่างโปรแกรมทำการเปิดแฟ้มเพื่อเขียน ("w") ชื่อ myfile และเขียนข้อความลงในแฟ้มคือ "This is my file for exception handling!!" ลงในแฟ้มดังกล่าว บรรทัดที่ 5 (except) เป็นคำสั่งเพื่อจัดการกับความผิดพลาดที่เกิดขึ้น โดยปกติการเปิดแฟ้มจะต้องทำการเชื่อมต่อ กับ Standard IO คือฮาร์ดดิสก์ ซึ่งเป็นอุปกรณ์ที่อาจจะเกิดข้อผิดพลาดขึ้นได้ ดังนั้นในโปรแกรมจึงใช้ คำสั่ง try ทำการครอบส่วนของโปรแกรมที่ติดต่อกับ IO เอาไว้ ถ้ามีข้อผิดพลาดเกิดขึ้น เช่น เปิดแฟ้ม ไม่ได้ มีโปรแกรมอื่นๆ ใช้งานอยู่ เป็นต้น โปรแกรมจะทำการคำสั่งในส่วน except IOError แทน (บรรทัดที่ 5) โดยพิมพ์ข้อความแสดงข้อผิดพลาดคือ "Error: can't find file or read data" แต่ถ้าโปรแกรมไม่มี ความผิดพลาดใดๆ เกิดขึ้น (เปิดแฟ้มและเขียนข้อมูลลงแฟ้มได้สำเร็จ) โปรแกรมจะทำงานต่อหลังคำสั่ง else (บรรทัดที่ 7) โดยพิมพ์ข้อความคือ "Written content in the file successfully" และปิดแฟ้มข้อมูล ในบรรทัดที่ 9 ตามลำดับ

จากตัวอย่างให้ทำการทดสอบอ่าน-เขียนแฟ้มใหม่อีกครั้ง โดยในครั้งนี้กำหนดสิทธิ์ให้อ่าน แฟ้มข้อมูลได้อย่างเดียว (บนระบบปฏิบัติการวินโดว์ส์) โดยคลิกขวาที่แฟ้มชื่อ myfile ⇒ เลือก Properties ⇒ ส่วนของ Attributes ให้ทำการคลิกเลือก read-only ⇒ เลือก Ok ดังรูปที่ 9.1



รูปที่ 9.1 แสดงการกำหนดสิทธิ์ชนิดอ่านได้อย่างเดียวให้กับแฟ้มชื่อ myfile

จากนั้นให้ทำการรันโปรแกรมที่ 9.1 อีกครั้ง ผลลัพธ์ที่ได้คือ



Error: can't find file or read data

ข้อผิดพลาดเกิดขึ้นจากโปรแกรมไม่สามารถเขียนข้อมูลลงไปในแฟ้มได้ เพราะแฟ้มถูกกำหนดสิทธิ์ให้อ่านได้อย่างเดียว เมื่อเกิดความผิดพลาดขึ้น โปรแกรมจะทำงานหลังคำสั่ง except IOError โดยพิมพ์ข้อความ “Error: can't find file or read data” ออกทางจอภาพ และไม่ทำคำสั่งที่อยู่หลัง else (ไม่จำเป็นต้องปิดแฟ้ม เพราะว่าโปรแกรมไม่สามารถเปิดแฟ้มได้)

สำหรับความผิดพลาดที่เกิดจากการป้อนข้อมูลทางแป้นพิมพ์ เช่น ผู้ใช้งานป้อนข้อมูลผิดประเภท ก็จะส่งผลทำให้เกิดข้อผิดพลาดได้ เช่น

```
>>> n = int(input("Please enter a number: "))
Please enter a number: 23.5      #invalid data type
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '23.5'
```

ตัวอย่างโปรแกรมที่ 9.2 แสดงการใช้ try...except กับการจัดการข้อผิดพลาดจากแป้นพิมพ์

Program Example 9.2: try...except with keyboard

```
1  # Try...except with keyboard
⇒2  while True:
⇒3      try:
⇒4          n = int(input("Please enter an integer: "))
⇒5          break
6      except ValueError:
⇒7          print("No valid integer! Please try again ...")
8  print("Great, you successfully entered an integer!")
```



```
Please enter an integer: 5.6
No valid integer! Please try again ...
Please enter an integer: 4
Great, you successfully entered an integer!
```

จากตัวอย่างโปรแกรมที่ 9.2 บรรทัดที่ 2 โปรแกรมทำการวนลูปแบบไม่รุบง เพราะเงื่อนไขที่ while ตรวจสอบเป็นจริงเสมอ บรรทัดที่ 3 โปรแกรมทำการตรวจสอบความผิดพลาดที่อาจจะเกิดขึ้นกับการป้อนข้อมูลจากแป้นพิมพ์โดยใช้คำสั่ง try บรรทัดที่ 4 โปรแกรมรับข้อมูลผ่านแป้นพิมพ์เป็นสตริง จนนั้นทำการแปลงเป็นเลขจำนวนเต็มด้วยคำสั่ง int ถ้าผู้ใช้งานป้อนข้อมูลเป็นเลขจำนวนเต็มโปรแกรมจะหยุดการทำงานของคำสั่ง while ด้วยคำสั่ง break (บรรทัดที่ 5) และกระโดดไปทำงานต่อในบรรทัดที่ 8 เพื่อพิมพ์ข้อความว่า “Great, you successfully entered an integer!” พร้อมกับจบการทำงาน แต่ถ้าผู้ใช้ป้อนข้อมูลชนิดอื่นๆ เช่น จำนวนจริง ตัวอักษร เป็นต้น โปรแกรมจะเกิดข้อผิดพลาดขึ้น ส่งผลให้โปรแกรมกระโดดไปทำงานหลังคำสั่ง except (บรรทัดที่ 6) ซึ่งประกาศไว้ว่าเป็นความผิดพลาดชนิด ValueError (ข้อมูลผิดประเภท) โปรแกรมจะสั่งพิมพ์ข้อความว่า “No valid integer! Please try again ...” (บรรทัดที่ 7) จากนั้นโปรแกรมจะกลับไปรับข้อมูลจากแป้นพิมพ์ใหม่ไปเรื่อยๆ จนกว่าผู้ใช้งานจะป้อนข้อมูลให้ถูกต้อง (เลขจำนวนเต็มเท่านั้น)

การตรวจสอบความผิดพลาดแบบไม่กำหนด Exceptions

ในบางกรณีผู้เขียนโปรแกรมไม่ทราบชนิดของความผิดพลาดที่อาจจะเกิดขึ้น หรือไม่ต้องการกำหนดรายละเอียดของความผิดพลาดที่จะเกิดขึ้นเหล่านั้น พร้อมอนุญาตให้ผู้เขียนโปรแกรมไม่จำเป็นต้องกำหนดชนิดของความผิดพลาดได้ โดยมีรูปแบบคำสั่งดังนี้

```
try:
    normal statement(s)
except:
    exception statement(s)
else:
    normal statement(s)
```

ตัวอย่างโปรแกรมที่ 9.3 แสดงการตรวจสอบข้อผิดพลาดที่ไม่กำหนด Exception

Program Example 9.3: undefined exception

```
1 # Try...except with no any exceptions
2 try:
3     fh = open("myfile", "w")
4     fh.write("This is my file for exception handling!!")
5 except:
6     print ("IO Error with File")
7 else:
```

```

8     print ("Written content in the file successfully")
9     fh.close()

```



IO Error with File

จากโปรแกรมที่ 9.3 เป็นโปรแกรมที่ทำหน้าที่เปิดแฟ้มเพื่อเขียนข้อมูล โดยกำหนดให้มีการตรวจสอบข้อผิดพลาดที่อาจเกิดขึ้นจากการเปิดแฟ้มด้วยคำสั่ง `try...except` แต่ไม่กำหนดชนิดความผิดพลาดอันใดอันหนึ่งไว้หลังคำสั่ง `except` (บรรทัดที่ 5) เมื่อโปรแกรมเกิดความผิดพลาดขึ้น โปรแกรมสั่งพิมพ์ข้อความ “IO Error with File” ออกมาก่อนแล้วนั้น ซึ่งการเขียนโปรแกรมที่ดีไม่ควรกระทำในลักษณะเช่นนี้ เพราะเมื่อโปรแกรมทำงานผิดพลาด ผู้เขียนโปรแกรมจะค้นหาสาเหตุของความผิดพลาดได้ยาก เพราะโปรแกรมแสดงข้อความแบบทั่วๆไปไม่ได้เจาะจงหรือเชื่อมโยงกับปัญหาจริงที่เกิดขึ้น

การตรวจสอบความผิดพลาดแบบหลาย Exceptions (Multiple exceptions)

Try...except อนุญาตให้ผู้เขียนโปรแกรมสามารถกำหนดเงื่อนไขการเกิดข้อผิดพลาดได้มากกว่า 1 ชนิดได้ภายใน `try` คำสั่งเดียว กัน และขณะเดียวกันนั่นจะมี `exception` เพียงอันเดียวเท่านั้นที่จะได้ถูกประมวลผล ซึ่งมี 2 รูปแบบดังนี้

แบบที่ 1 กำหนด Exception แยกจากกัน

```

try:
    normal statement(s)
except Exception 1:
    exception statement(s)
except Exception 2:
    exception statement(s)
    .....
except Exception n:

```

แบบที่ 2 กำหนด Exception อุยกายได้คำสั่ง `except` เดียว กัน

```

try:
    normal statement(s)
except(Exception1[, Exception2,...ExceptionN]):
    exception statement(s)

```

ตัวอย่างโปรแกรมที่ 9.4 แสดงการใช้งาน Multiple exceptions ทั้ง 2 แบบ

Program Example 9.4: multiple exceptions

```

1 # Try...except with multiple exceptions
2 import sys
3 #This code for separating errors
4 try:
5     f = open('integers.txt')
6     s = f.readline()
7     i = int(s.strip())
⇒8 except IOError:
8     print("I/O error")
⇒10 except ValueError:
9     print("No valid integer in line.")
⇒12 except:
10    print("Unexpected error")
11
12 #This code for combining errors
13 try:
14     fh = open("testfile", "w")
15     fh.write("This is my file for exception handling!!")
⇒19 except(IOError, ValueError, SystemError):
16     print("Error: can't find file or read data")
17 else:
18     print("Written content in the file successfully")
19 fh.close()

```



No valid integer in line.
Written content in the file successfully

จากโปรแกรมที่ 9.4 แสดงการใช้ `try...except` จัดการกับ exception หลายประเภท ในส่วนแรกของโปรแกรม (บรรทัดที่ 4 - 13) ทดสอบโดยการเปิดแฟ้มชื่อ "integer.txt" แบบอ่านได้อย่างเดียว จากนั้นใช้ฟังction `readline()` เพื่ออ่านข้อมูลในแฟ้มจำนวน 1 บรรทัดเก็บไว้ในตัวแปร `s` บรรทัดที่ 6 โปรแกรมจะตัดข้อความว่างออกจากสตริงด้วยฟังction `strip()` จากนั้นสตริงจะถูกแปลงให้เป็นข้อมูลชนิดจำนวนเต็มด้วยฟังction `int()` ความผิดพลาดชนิด `IOError` (บรรทัดที่ 8) เกิดขึ้นจากโปรแกรมไม่สามารถเปิดแฟ้มชื่อ `integers.txt` เพื่ออ่านได้ หรือไม่มีแฟ้มดังกล่าวอยู่ในไดเรคทรอรีปัจจุบัน ความผิดพลาดชนิด `ValueError` (บรรทัดที่ 10) เกิดขึ้นจากแฟ้ม `integers.txt` ไม่มีข้อมูลใดๆ หรือข้อมูลไม่ใช่ตัวเลข (เกิดจากคำสั่ง `int(s.strip())`) แต่ถ้าข้อผิดพลาดไม่อยู่ใน 2 ชนิดแรก โปรแกรมจะทำงานในส่วน `except:` (บรรทัดที่ 12) โดยพิมพ์ข้อความว่า "Unexpected error"

สำหรับส่วนที่สองของโปรแกรม 9.4 (บรรทัดที่ 16 - 23) โปรแกรมทำการเปิดแฟ้มชื่อ `testfile.txt` แบบเขียนได้ ("w") เมื่อโปรแกรมเริ่มอ่านเขียนแฟ้มและเกิดข้อผิดพลาดขึ้น โปรแกรมจะทำงานในบรรทัดที่ 19 ซึ่งได้รวมเอาความผิดพลาดหลายชนิดเข้าไว้ภายใน `except` เดียวกันคือ `IOError, ValueError, SystemError` เมื่อเกิดข้อผิดพลาดอย่างใดอย่างหนึ่งขึ้นใน 3 ชนิดข้างต้น โปรแกรมจะพิมพ์ข้อความว่า "Error: can't find file or read data"

การตรวจสอบความผิดพลาดแบบ try...else..finally

จากตัวอย่างที่กล่าวมาแล้วข้างต้น เมื่อโปรแกรมเกิดข้อผิดพลาดขึ้นโปรแกรมจะทำงานในส่วนหลังของคำสั่ง except แต่ถ้าโปรแกรมทำงานปกติไม่มีข้อผิดพลาดใดๆ จะประมวลผลหลังคำสั่ง else (ถ้าผู้เขียนโปรแกรมประกาศคำสั่ง else ไว้ใน try...except) แต่สำหรับคำสั่ง try...finally จะมีการใช้งานที่แตกต่างไปเล็กน้อยคือ try...finally จะทำงานทุกๆ ครั้ง แม้ว่าโปรแกรมจะเกิดข้อผิดพลาด หรือไม่ก็ตาม ซึ่งมีรูปแบบคำสั่งดังนี้

```
try:
    normal statement(s)
finally:
    finally statement(s)
```

สำหรับการใช้ try...finally มีข้อกำหนดที่ควรทราบดังนี้

- สามารถใช้คำสั่ง except ร่วมกับคำสั่ง finally ได้ แต่ควรระวังเรื่องของความหมายในการแสดงผล และจำไว้เสมอว่าเมื่อโปรแกรมทำงานผิดพลาดจะทำงานหลังคำสั่ง except และ finally แต่ถ้าโปรแกรมทำงานปกติ โปรแกรมจะทำคำสั่งหลัง finally
- สามารถใช้คำสั่ง except ร่วมกับ finally และ else ได้ (ไม่แนะนำให้ทำ) แต่ต้องเรียงลำดับให้ถูกต้อง โดยเรียงลำดับคำสั่ง ดังนี้

```
try:
    normal statement(s)
except:
    exception statement(s)
else:
    else statement(s)
finally:
    finally statement(s)
```

ตัวอย่างโปรแกรมที่ 9.5 แสดงตัวอย่างการใช้งาน try...except, else และ finally ร่วมกัน

Program Example 9.5: try...except, else and finally	
--	--

```

1   # Try...except, else and finally
2   try:
3       fh = open("myfile", "w")
4       fh.write("This is my file for exception handling!!")
5   except:
6       print("IO Error with File")
7   else:
8       print("Written content in the file successfully")
9   finally:
10      print("This file is closed completely")
11      fh.close()
```

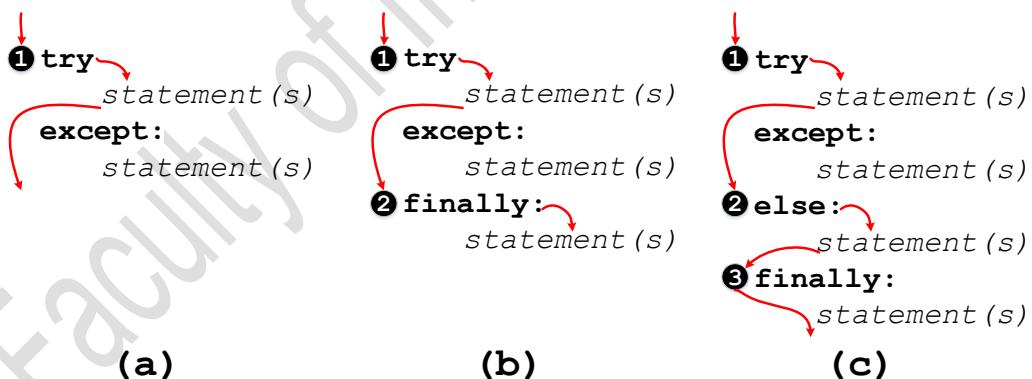


Written content in the file successfully
This file is closed completely



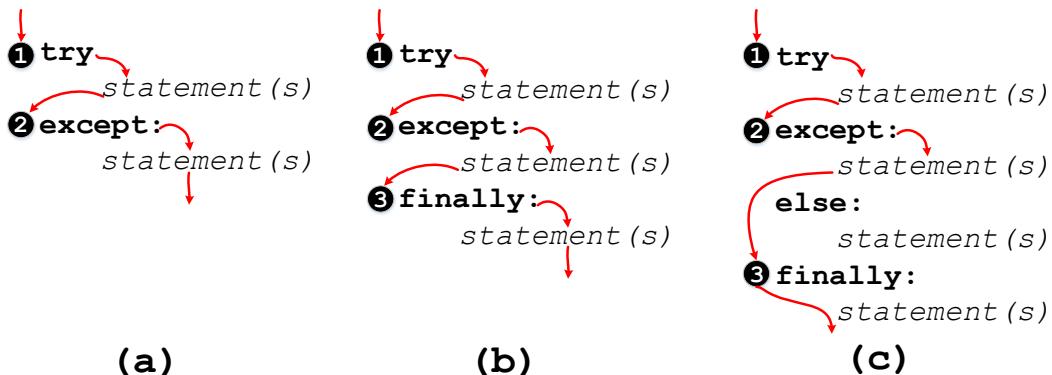
```
IO Error with File
This file is closed completely
Traceback (most recent call last):
  File "C:/Python33/exam9_5.py", line 11, in <module>
    fh.close()
NameError: name 'fh' is not defined
```

จากโปรแกรมที่ 9.5 บรรทัดที่ 3 โปรแกรมทำการเปิดแฟ้มแบบอ่านเขียนได้ ("w") ถ้าโปรแกรมสามารถเปิดแฟ้มและเขียนข้อมูลลงแฟ้มได้ (บรรทัดที่ 4) โดยไม่มีข้อผิดพลาดใดๆ เกิดขึ้น โปรแกรมจะประมวลผลหลังคำสั่ง else (บรรทัดที่ 7) โดยพิมพ์ข้อความว่า "Written content in the file successfully" จากนั้นโปรแกรมจะทำการfinally ต่อ (บรรทัดที่ 9) โดยพิมพ์ข้อความว่า "This file is closed completely" พร้อมกับปิดแฟ้ม (บรรทัดที่ 11) และจบการทำงานโดยสมบูรณ์ แต่ถ้าโปรแกรมไม่สามารถอ่านหรือเขียนแฟ้มอย่างได้อย่างหนึ่ง โปรแกรมจะทำงานหลังคำสั่ง except (บรรทัดที่ 5) โดยพิมพ์ข้อความว่า "IO Error with File" เมื่อทำการfinally ดังกล่าวเสร็จ โปรแกรมจะกระโดดไปทำงานหลังคำสั่ง finally โดยพิมพ์ข้อความว่า "This file is closed completely" พร้อมกับปิดแฟ้ม แต่โปรแกรมจะเกิดข้อผิดพลาดคือ "NameError: name 'fh' is not defined" เนื่องจากโปรแกรมตรงส่วน finally พยายามจะปิดแฟ้มซึ่งไม่ได้เปิดไว้ (เพราะเกิดข้อผิดพลาดก่อนเปิดแฟ้มได้ในส่วนของคำสั่ง try) ทำให้ไฟล์ของโปรแกรมแจ้งเตือนว่าแฟ้ม (fh) ดังกล่าวไม่เคยมีการประกาศไว้



รูปที่ 9.2 แสดงการใช้ try...except, else และ finally (กรณีไม่เกิดความผิดพลาด)

จากรูปที่ 9.2 แสดงการใช้ try...except, else และ finally เพื่อดักจับข้อผิดพลาดที่อาจจะเกิดขึ้นกับโปรแกรม (a) โปรแกรมประมวลผลใน try และไม่เกิดความผิดพลาด โปรแกรมจะไม่ประมวลผลคำสั่ง except (b) โปรแกรมประมวลผลใน try และไม่เกิดความผิดพลาด โปรแกรมจะประมวลผลคำสั่ง finally และ (c) โปรแกรมประมวลผลใน try และไม่เกิดความผิดพลาด โปรแกรมจะประมวลผลทั้งคำสั่ง else และ finally



รูปที่ 9.3 แสดงการใช้ try...except, else และ finally (กรณีเกิดความผิดพลาด)

จากรูปที่ 9.3 (a) โปรแกรมประมวลผลใน try และเกิดความผิดพลาด โปรแกรมจะประมวลผล
คำสั่ง except (b) โปรแกรมประมวลผลใน try และเกิดความผิดพลาด โปรแกรมจะประมวลผลทั้งคำสั่ง
except และ finally (c) โปรแกรมประมวลผลใน try และเกิดความผิดพลาด โปรแกรมจะประมวลผล
คำสั่ง execpt และ finally ยกเว้นคำสั่ง else

การเขียนโปรแกรมด้วยการใช้ `try...except` ที่ดีไม่ควรใช้คำสั่งที่มีการทำงานซ้ำกันในโปรแกรม เช่น ถ้าผู้เขียนโปรแกรมใช้ `finally` แล้ว ไม่ควรใช้ `else` อีก หรือถ้าใช้คำสั่ง `except` ร่วมกับ `else` แล้วก็ ไม่ควรใช้ `finally` เช่นเดียวกัน จากตัวอย่างโปรแกรมที่ 9.6 แสดงการใช้ `try...except` ที่ง่ายต่อการตีความหมายของโปรแกรม

ตัวอย่างโปรแกรมที่ 9.6 แสดงตัวอย่างการใช้งาน try...except ที่ดี

Program Example 9.6: good try...except

```
1      # How to use good try...except
2      try:
3          fh = open("myfile", "w")
4          try:
5              fh.write("This is my test file for exception
6 handling!!")
7              finally:
8                  print("Going to close the file")
9                  fh.close()
10             except IOError:
11                 print("Error: can't find file or read data")
```



```
Going to close the file  
Error: can't find file or read data
```

จากโปรแกรมตัวอย่างที่ 9.6 แสดงการใช้ try ซ้อน try โดยคำสั่ง try ลำดับที่ 1 (บรรทัดที่ 2) จะตรวจสอบการเปิดแฟ้มเพื่ออ่านเขียน ถ้าการเปิดแฟ้มเกิดข้อผิดพลาดขึ้น (บรรทัดที่ 3) โปรแกรมจะไปทำงานที่หลังคำสั่ง except IOError (บรรทัดที่ 9) โดยพิมพ์ข้อความว่า “Error: can't find file or read data” และถ้าโปรแกรมทำงานเป็นปกติ จะทำงานในคำสั่งลำดับถัดไป (บรรทัดที่ 5) คือการเขียนข้อความ

ลงแฟ้มมีข้อความว่า “This is my test file for exception handling!!” เมื่อข้อมูลถูกเขียนเสร็จเรียบร้อยแล้ว โปรแกรมจะพิมพ์ข้อความว่า “Going to close the file” (บรรทัดที่ 7) พร้อมกับปิดแฟ้มข้อมูลและยุติการทำงานของโปรแกรม สังเกตว่า try...except ที่อยู่ด้านนอกจะจัดการเกี่ยวกับการเปิดแฟ้ม ส่วน try...finally ที่อยู่ด้านในจะจัดการความผิดพลาดเกี่ยวกับการเขียนแฟ้ม โดย finally จะทำหน้าที่ปิดแฟ้มเสมอ



Note: try...finally จะทำงานตลอดเวลา แม้ว่าการทำงานของโปรแกรมจะผิดปกติหรือไม่ก็ตาม ดังนั้น ถ้าโปรแกรมเกิดข้อผิดพลาดขึ้น อาจจะไม่ทราบว่าเกิดขึ้นหรือไม่อย่างไร เพราะโปรแกรมจะเข้าไปทำงานในส่วน finally เสมอ เช่น ในโปรแกรมที่ 9.6 ผู้เขียนโปรแกรมจะไม่ทราบเลยว่ามีข้อผิดพลาดที่เกิดขึ้นขณะเขียนข้อความลงแฟ้มหรือไม่ เพราะโปรแกรมจะทำคำสั่ง finally เสมอ

อาร์กิวเมนต์ของ Exception

ไฟชอนอนุญาตให้ผู้เขียนโปรแกรมสามารถสร้างอาร์กิวเมนต์เพื่อรับข้อมูลเกี่ยวกับความผิดพลาดหรือปัญหาที่เกิดขึ้นจากไฟชอนคอมไพล์เลอร์ได้ โดยมีรูปแบบคำสั่งดังนี้

```

try:
    normal statement(s)
except ExceptionType as Args:
    exception statement(s)

```

จากรูปแบบคำสั่งข้างบน ExceptionType as คือ ชนิดของความผิดปกติที่เกิดขึ้นในโปรแกรม ซึ่งแสดงในรูปที่ 9.4 และ Args คือ ตัวแปรอาร์กิวเมนต์ที่ผู้เขียนโปรแกรมกำหนดขึ้นเอง เพื่อสำเนาข้อมูลความผิดปกติจากออบเจกต์ ExceptionType ตัวอย่างการใช้งานดังโปรแกรมที่ 9.7

ตัวอย่างโปรแกรมที่ 9.7 แสดงตัวอย่างการใช้งานอาร์กิวเมนต์ของ exception

Program Example 9.7: except arguments

```

1 # Argument of Exception
2 # Define a function here.
3 def temp_convert(var):
4     try:
5         return int(var)
6     except ValueError as Args:
7         print ("Argument doesn't contain numbers\n", Args.args)
8 # Call above function here.
9 temp_convert("xyz");

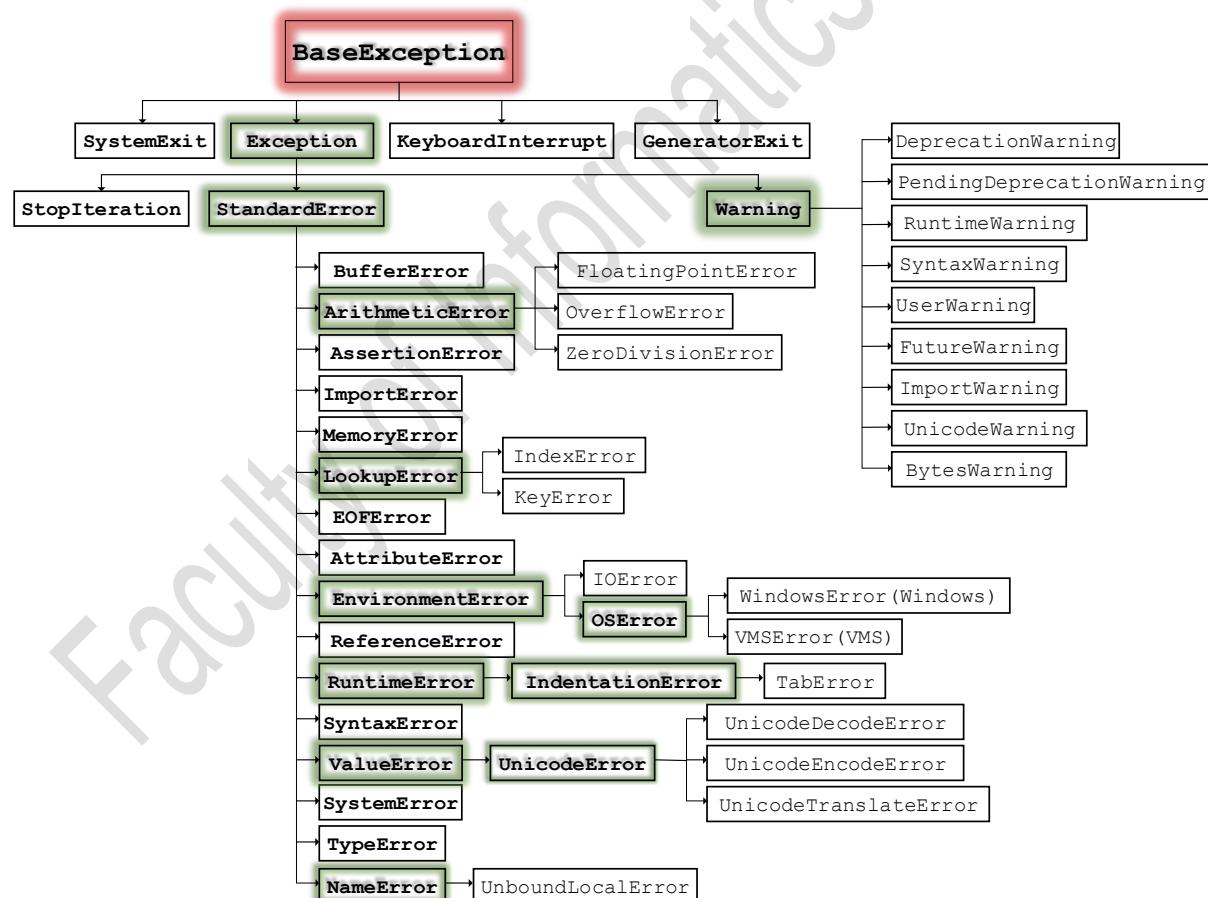
```



Argument does not contain numbers
("invalid literal for int() with base 10: 'xyz'")

จากโปรแกรมที่ 9.7 แสดงการสร้างอาร์กิวเมนต์เพื่อรับข้อมูลความผิดพลาดจาก Python คอมไพล์เลอร์ บรรทัดที่ 3 โปรแกรมประกาศพังชันชื่อ temp_convert ทำหน้าที่แปลงอ้อปเจ็กไดๆ ให้เป็นชุดจำนวนเต็ม (บรรทัดที่ 5) โดยมีพารามิเตอร์ 1 ตัวคือ var บรรทัดที่ 4 โปรแกรมทำการตรวจสอบความผิดพลาดไว้ ในการนี้ที่ไม่สามารถแปลงข้อมูลได้สำเร็จ โปรแกรมจะทำงานในบรรทัดที่ 6 ซึ่งทำการสำเนาความผิดพลาดจากตัวแปรระบบคือ ValueError ไปเป็น Args ด้วยคำสั่ง as เมื่อข้อผิดพลาดเกิดขึ้น เช่น อ้อปเจ็ก var ไม่เป็นจำนวนเต็ม โปรแกรมจะพิมพ์ข้อความว่า "Argument doesn't contain numbers" พร้อมกับความผิดพลาดที่ถูกสำเนาเก็บไว้ในตัวแปร Args.args (เมื่อต้องการแสดงข้อมูลผิดพลาดในอาร์กิวเมนต์ Args ให้ใช้ .attribute เช่น Args.args)

บรรทัดที่ 9 โปรแกรมทำการเรียกพังชัน temp_convert พร้อมค่าคงที่สตริงเป็นอาร์กิวเมนต์ ให้กับพังชัน ผลปรากฏว่าพังชันแสดงข้อความ Argument does not contain numbers ("invalid literal for int() with base 10: 'xyz'") เพราะอาร์กิวเมนต์ที่ส่งเข้าไปให้พังชันผิดประเภท (ต้องการสตริงที่เป็นตัวเลข ไม่ใช่ตัวอักษร) นั่นเอง



รูปที่ 9.4 แสดงประเภทของ Exceptions

จากข้อปฏิทักษ์ 9.4 คลาส Exception เป็นคลาสหลักที่สำคัญ ครอบคลุมความผิดพลาดที่เกิดขึ้นเกือบทั้งหมด โดยคลาส StandardError เป็นคลาสลูกที่สืบทอดคุณสมบัติมาจากคลาส Exception คลาสนี้ทำหน้าที่ดูแลเกี่ยวกับความผิดพลาดพื้นฐานที่พบเจอบ่อยๆ ในการเขียนโปรแกรม เช่น ArithmeticError ซึ่งเป็นความผิดพลาดที่เกี่ยวกับการประมวลผลทางคณิตศาสตร์ MemoryError เป็นความผิดพลาดเกี่ยวกับการอ้างถึงหน่วยความจำ RuntimeError เป็นความผิดพลาดที่เกิดขึ้นขณะโปรแกรมกำลังทำงาน เช่น คันหาแฟ้มไม่พบ ชนิดตัวแปรผิดพลาด เป็นต้น SyntaxError คือความผิดพลาดของการเขียนโปรแกรมผิดไวยกรณ์ และ SystemError คือความผิดพลาดเกี่ยวกับระบบ เช่น พื้นที่หน่วยความจำในดิสก์เต็ม ระบบเครื่อข่ายไม่สามารถเชื่อมต่อได้ เป็นต้น สำหรับชนิดและความหมายของความผิดพลาดต่างๆ สามารถอ่านเพิ่มเติมได้จาก Python Library (<http://docs.python.org/3.1/library/exceptions.html>)

การตรวจจับความผิดพลาดด้วยคำสั่ง Raising

Raising คือ คำสั่งที่ใช้บนอนุญาตให้ผู้เขียนโปรแกรมสามารถสร้างข้อความแจ้งเตือนความผิดพลาดที่อาจจะเกิดขึ้นได้ด้วยตนเอง ซึ่งคำสั่งดังกล่าวจะทำหน้าที่ 2 อย่างคือ สร้าง Exception object เพื่อใช้สำหรับอ้างอิงกับความผิดพลาดที่ใช้บนสร้างไว้ (Built-in exception) และการโอนความผิดพลาดไปยังโปรแกรมที่เรียกใช้งาน สำหรับการใช้งาน raising มีรูปแบบคือ

```
raise [Exception [, args [, traceback]]]
```

raise คือคำสั่งตรวจจับความผิดพลาด Exception คือ ชนิดของความผิดพลาดที่เกิดขึ้น เช่น NameError, IOError เป็นต้น ซึ่งอาร์กิวเมนต์ดังกล่าวจะกำหนดหรือไม่ก็ได้ (เพราะอยู่ในเครื่องหมาย [...]]) args เป็นอาร์กิวเมนต์ของ exception ถ้าไม่ได้กำหนดไว้ Python จะถือว่าเป็นค่าเป็น None อาร์กิวเมนต์ traceback เป็นอ็อปชันที่ใช้สำหรับแสดงความผิดพลาดแบบย้อนกลับ (ต้นเหตุของปัญหา) สำหรับตัวอย่างการใช้งาน raising แสดงต่อโปรแกรมที่ 9.8, 9.9

ตัวอย่างโปรแกรมที่ 9.8 แสดงตัวอย่างการใช้งาน raise

Program Example 9.8: raise

```

1 # Raising exception example 1
⇒2 def functionName(level):
⇒3     if level < 1:
⇒4         raise ("Invalid level!", level)
⇒5         print("The code would not be executed")
⇒6         print("Good Bye!")
7
⇒8 functionName(5)
⇒9 functionName(0)

```



OUTPUT

```

Good Bye!
Traceback (most recent call last):
  File "C:\Python34\exam9_8.py", line 9, in <module>
    functionName(0)
  File "C:\Python34\exam9_8.py", line 4, in functionName
    raise ("Invalid level!", level)
TypeError: exceptions must derive from BaseException

```

จากโปรแกรมตัวอย่างที่ 9.8 แสดงการใช้คำสั่ง raising เพื่อสร้างข้อความแจ้งเตือนความผิดพลาดที่ผู้ใช้กำหนดขึ้นเอง บรรทัดที่ 2 โปรแกรมประกาศฟังชันชี้ว่า functionName ทำหน้าที่ตรวจสอบระดับสิทธิ์ในการประมวลผล ถ้าระดับสิทธิ์ (level) น้อยกว่า 1 จะไม่อนุญาตให้ประมวลผล พังชันดังกล่าวมีพารามิเตอร์ 1 ตัวคือ level (ระดับสิทธิ์การประมวลผล) บรรทัดที่ 3 โปรแกรมใช้ if ในการตรวจสอบเงื่อนไขว่าถ้า level น้อยกว่า 1 หรือไม่ ถ้าใช้โปรแกรมจะสร้างข้อความแจ้งเตือนความผิดพลาดใหม่โดยใช้คำสั่ง raise (บรรทัดที่ 4) โดยพิมพ์ข้อความว่า "Invalid level!" พร้อมกับค่าในตัวแปร level ไปให้กับผู้เรียกใช้งานและจบโปรแกรมทันทีโดยไม่ประมวลผลคำสั่งในบรรทัดที่ 5 ซึ่งพิมพ์ข้อความแจ้งเตือนว่า "The code would not be executed" แต่ถ้า level มีค่ามากกว่า 1 โปรแกรมจะพิมพ์ข้อความว่า "Good Bye!" และยุติการทำงานของโปรแกรม

บรรทัดที่ 8 โปรแกรมเรียก functionName พร้อมค่าคงที่เป็นอาร์กิวเม้นต์มีค่าเท่ากับ 5 ผลลัพธ์ที่ได้จาก functionName คือข้อความว่า "Good Bye!" บรรทัดที่ 9 ทดสอบเรียกฟังชันเดิมอีกครั้ง พร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์มีค่าเท่ากับ 0 ผลลัพธ์ที่ได้คือ ข้อความผิดพลาดที่เกิดจากคำสั่ง raise พร้อมกับข้อผิดพลาดที่ไฟล์สามารถตรวจสอบได้ ดังแสดงในตัวอย่างข้างบน

ตัวอย่างโปรแกรมที่ 9.9 แสดงตัวอย่างการใช้คำสั่ง raise อีกแบบหนึ่ง

Program Example 9.9: other rise exam

```

1 # Raising exception example 2
2 try:
3     f = open("myfile","w")
4     f.write("This is my test file for exception handling!!")
5 except(IOError, ValueError):
6     print("An I/O error or a ValueError occurred")
7     raise OSError("Permission denied")
8 except:
9     print("An unexpected error occurred")
10    raise
11 else:
12     f.close()
13     print("Myfile was closed!!")
14 finally:
15     print("Program has finished")

```



OUTPUT

```

# While Myfile attribute can read-write
Myfile was closed!!
Program has finished

```

```

# While Myfile attribute can read-only
An I/O error or a ValueError occurred
Program has finished
Traceback (most recent call last):
  File "C:/Python33/exam9_9.py", line 4, in <module>
    f = open("myfile", "w")
PermissionError: [Errno 13] Permission denied: 'myfile'

During handling of the above exception, another exception
occurred:

Traceback (most recent call last):
  File "C:/Python33/exam9_9.py", line 8, in <module>
    raise OSError("Permission denied")
OSError: Permission denied

```

จากโปรแกรมตัวอย่างที่ 9.9 แสดงการทำงานของคำสั่ง `raise` อีกรูปแบบหนึ่ง เริ่มต้นจากบรรทัดที่ 3 โปรแกรมทำการเปิดแฟ้มชื่อ `myfile` เพื่อเขียน ("w") ถ้าโปรแกรมสามารถเขียนข้อความว่า "This is my test file for exception handling!!" ลงแฟ้มได้สำเร็จ (บรรทัดที่ 4) โปรแกรมจะไปประมวลผลคำสั่ง `else` ในบรรทัดที่ 12 คือปิดแฟ้มพร้อมกับพิมพ์ข้อความว่า "Myfile was closed!!" ออกทางจอภาพ (บรรทัดที่ 13) จากนั้นโปรแกรมจะประมวลผลคำสั่ง `finally` ต่อในบรรทัดที่ 14 โดยพิมพ์ข้อความว่า "Program has finished" และยุติการทำงานของโปรแกรม แต่ถ้าโปรแกรมไม่สามารถเขียนข้อมูลลงแฟ้ม `myfile` ได้ (ทดสอบโดยการคลิกขวาที่แฟ้ม `myfile` และเลือก attribute เป็น `read-only`) โปรแกรมจะเกิดข้อผิดพลาดขึ้น

ถ้าเกิดความผิดพลาดเป็นชนิด `IOError` และ `ValueError` (บรรทัดที่ 5) โปรแกรมจะไปประมวลผลหลังคำสั่ง `except (IOError, ValueError)` ในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า "An I/O error or a ValueError occurred" ต่อจากนั้นบรรทัดที่ 7 โปรแกรมจะทำการประมวลผลคำสั่ง `raise` โดยกรองเอาเฉพาะความผิดพลาดชนิด `OSError` ไว้ ถ้าโปรแกรมเกิดความผิดพลาดตามชนิดที่ระบุไว้ (`OSError`) จะพิมพ์ข้อความว่า "Permission denied" และจะประมวลผลคำสั่ง `finally` เป็นอันดับสุดท้าย

แต่ถ้าความผิดพลาดที่เกิดขึ้นไม่ใช่ทั้ง `IOError` และ `ValueError` โปรแกรมจะประมวลผลที่คำสั่ง `except` (บรรทัดที่ 8) โดยพิมพ์ข้อความว่า "An unexpected error occurred" (บรรทัดที่ 9) พร้อมกับส่งให้ไฟลอนแสดงความผิดพลาดที่เกิดขึ้นจริงๆ ในระบบด้วยคำสั่ง `raise` (บรรทัดที่ 10) อีกครั้ง สุดท้ายโปรแกรมจะทำงานหลังคำสั่ง `finally` (ทำงานทุกครั้งแม้ว่าจะเกิดหรือไม่เกิดข้อผิดพลาดใดๆ ก็ตาม) โดยไม่ประมวลผลคำสั่ง `else`

ตัวอย่างโปรแกรมที่ 9.10 แสดงการตรวจจับความผิดพลาดที่เกิดจากจำนวนเต็มที่น้อยกว่า 0

Program Example 9.10: detecting zero number

```

1 | # Raising exception example 3
2 | def f(x):

```

```

3         return g(x) + 1
4 def g(x):
5     if x < 0: raise ValueError, ("I can't settle with a
negative number."))
6     else: return 5
7 try:
8     print (f(3))
9     print (f(-5))
10 except ValueError:
11     print ("That value was invalid.")

```


OUTPUT

6

```

Traceback (most recent call last):
  File "C:/Python33/exam9_10.py", line 11, in <module>
    print (f(-5))
  File "C:/Python33/exam9_10.py", line 3, in f
    return g(x) + 1
  File "C:/Python33/exam9_10.py", line 6, in g
    if x < 0: raise ValueError, ("I can't settle with a
negative number."))
TypeError: exceptions must derive from BaseException

```

การสร้าง Exception ขึ้นมาใช้งานเอง (User-defined exceptions)

ไฟชอนอนุญาตให้ผู้เขียนโปรแกรมสามารถสร้าง exception ขึ้นมาใช้งานเองได้ โดยการสืบทอดคุณสมบัติมาจากคลาสที่เป็น built-in มาตรฐาน (standard built-in exceptions) ตัวอย่างเช่น เมื่อผู้เขียนโปรแกรมเขียนโปรแกรมเกี่ยวกับระบบเครือข่าย (Networking) และต้องการสร้าง exception เกี่ยวกับความผิดพลาดเกี่ยวกับการสื่อสาร ผู้เขียนโปรแกรมสามารถสืบทอด exception จากคลาส RuntimeError ซึ่งมีรูปแบบการสืบทอดดังนี้

```

class NetworkError(RuntimeError):
    def __init__(self, arg):
        self.args = arg

```

สำหรับการใช้งาน class NetworkError มีรูปแบบคือ

```

try:
    raise NetworkError("Bad hostname")
except NetworkError as e:
    print(e.args)

```


OUTPUT

```
('B', 'a', 'd', ' ', 'h', 'o', 's', 't', 'n', 'a', 'm', 'e')
```

จากตัวอย่างโปรแกรมข้างบน ใช้คำสั่ง `raise` สำหรับตรวจสอบและบังคับให้โปรแกรมพิมพ์ข้อความ “Bad hostname” ขณะโปรแกรมทำงานผิดพลาด และใช้ตัวแปร `e` ในการสำเนาข้อมูลที่เกิดความผิดพลาดมาแสดงผล

3. การยืนยันในสมมติฐาน (Assertions)

Assertions คือ การทดสอบและยืนยันสมมติฐานเกี่ยวกับโปรแกรมที่ได้เขียนไว้ ยกตัวอย่างเช่น กรณีที่มีเงื่อนไขในโปรแกรมที่ผู้เขียนมั่นใจว่าจะไม่มีโอกาสเป็นเท็จได้ (`False`) ผู้เขียนโปรแกรมควรจะ วาง assertion เป็นจริง (True) เอาไว้ ถ้าระหว่างการพัฒนาหรือใช้งานโปรแกรมแล้วเกิดความผิดพลาดที่เกิดจาก Assertion error แสดงว่าโปรแกรมตรงตามกำหนดดังกล่าวมีการทำงานที่ผิดพลาด เกิดขึ้น จำเป็นต้องแก้ไขให้ถูกต้อง หรือถ้าจะพูดให้สั้นๆ ง่ายๆ คือ การกรอง (`Filter`) ความถูกต้องของโปรแกรมในส่วนที่ผู้เขียนโปรแกรมกำหนดไว้ก่อนแล้ว การวางแผนของ assertion ไว้ในโปรแกรมโดยปกติจะว่างไว้ที่จุดเริ่มต้นของฟังชันเพื่อตรวจสอบอินพุตพารามิเตอร์ และวางไว้หลังจากที่เรียกฟังชัน เพื่อใช้สำหรับตรวจสอบเอกสารพุตที่ส่งกลับมา assertion มีรูปแบบคำสั่งดังนี้

`assert Expression[, Arguments]`

`assert` คือ คำสั่งที่ใช้ตรวจสอบสมมติฐาน, `Expression` คือ เงื่อนไขที่ต้องการตรวจสอบ และ `Arguments` คือ ข้อมูลเกี่ยวกับความผิดพลาด

เมื่อ Python เปลี่ยนคำสั่งมายังบรรทัดที่วาง `assert` เอาไว้ Python จะตรวจสอบเงื่อนไขที่กำหนดไว้ว่าเป็นจริงหรือไม่ ถ้าเป็นจริง Python จะประมวลผลคำสั่งตัดไป เมื่อไม่มีอะไรเกิดขึ้น (เมื่อคำสั่ง pass) แต่ถ้าเงื่อนไขเป็นเท็จ Python จะสร้างความผิดพลาด `AssertionError` อกมาให้ผู้เขียนโปรแกรมได้ทราบทันที โดยโอนความผิดพลาดเก็บไว้ใน `Arguments` โปรแกรมเมอร์สามารถจับความผิดพลาดดังกล่าวด้วยคำสั่ง `try...except` ได้ แต่ถ้าไม่ได้สร้างคำสั่งตรวจสอบไว้ เมื่อเกิดข้อผิดพลาดจาก assert โปรแกรมจะหยุดทำงาน และจะแสดงผลความผิดพลาดแบบย้อนกลับ (trackback) ให้ผู้เขียนโปรแกรมได้ทราบ

ข้อกำหนดในการใช้ assert

- วางคำสั่ง assert ไว้ในบรรทัดที่โปรแกรมเมอร์คิดว่าจะเป็นจริง (`True`) เสมอ
- ถ้าเงื่อนไขเป็นเท็จขณะทำงาน assert จะสร้าง `AssertionError` อกมา
- โปรแกรมเมอร์สามารถ disable การตรวจสอบได้
- โดยทั่วไปการใช้ assert จะกระทำในช่วงการพัฒนาโปรแกรม แต่หลังจากที่โปรแกรมพัฒนาได้สมบูรณ์หรือปลดภัยแล้ว ควร disable คำสั่ง assert ด้วย

ประโยชน์ของ assert

- ใช้เพื่อตรวจสอบสมมติฐานหรือคัดกรองความผิดพลาดของผู้เขียนโปรแกรม
- ช่วยในการหาข้อผิดพลาดของโปรแกรม (bug)
- เพิ่มความมั่นใจว่า โปรแกรมที่ใช้งานจะไม่มีข้อผิดพลาดหรือมีข้อผิดพลาดน้อยที่สุด
- ใช้งานได้สะดวกกว่า try...except เนื่องจากสามารถ disable การตรวจสอบของ assert ได้ในขณะที่โปรแกรมทำงาน (Run-time) โดยไม่ลดทอนประสิทธิภาพการทำงานของ การประมวลผลลง

ตำแหน่งที่นิยมวาง assert ไว้

- ในฟังชัน (ด้านบนสุดของฟังชัน) เพื่อตรวจสอบตัวแปรอินพุต
- หลังจากกลับมาจากการเรียกใช้งานฟังชัน เพื่อตรวจสอบค่าที่ส่งกลับจากฟังชัน
- วางแผนตำแหน่งของ comment ในโปรแกรม
- วางไว้หลังคำสั่งที่ไม่มีคำสั่ง default เช่น หลัง while, for เป็นต้น
- วาง assert ไว้ในลูปหรือเงื่อนไขที่ไม่มีโอกาสจะเกิดขึ้น (อาจจะไม่เกิดขึ้นในปัจจุบัน แต่อาจจะเกิดขึ้นได้ในอนาคต)

คำแนะนำเพิ่มเติมอีก ๔ เกี่ยวกับการใช้งาน assert

- Expression ของ assert ต้องไม่มีผลกระทบกับการทำงานในส่วนอื่นๆ ของโปรแกรม (side effect)
- ให้เพิ่ม assert ขณะกำลังเขียนโปรแกรม ไม่ควรเขียนหลังจากเขียนโปรแกรมเสร็จแล้ว (เพราะข้อสองสัยอาจจะหายไปด้วยเมื่อเวลาผ่านไป)
- ใช้ assert ได้มากเท่าที่ต้องการ ไม่ต้องกลัวว่าจะมากเกินไป ขอให้สามารถตรวจสอบ ความผิดพลาดของโปรแกรมได้มากที่สุด (เป้าหมายคือ โปรแกรมไม่มี bug เลย)
- ไม่ต้องห่วงเรื่องประสิทธิภาพการทำงานที่จะลดลง เนื่องจากสามารถ disable คำสั่ง assert ได้
- Exception ใช้กับสิ่งที่คิดว่าจะเกิดความผิดพลาด เมื่อความผิดพลาดเกิดขึ้นแล้ว จะต้อง มีวิธีแก้ไข แต่ assertion ใช้กับสิ่งที่ต้องไม่เกิดขึ้น ถ้าเกิดขึ้นแสดงว่าผิดพลาดค่อนข้าง ร้ายแรง
- Exception เป็นการเสริมให้โปรแกรมมีความคงทน ไม่ล้มเหลวง่ายๆ (Robustness) แต่ assert เป็นการสร้างความเชื่อมั่น (Reliability) ให้กับโปรแกรมที่เขียนขึ้น

ตัวอย่างโปรแกรมที่ 9.11 การตรวจสอบ precondition ไว้ในฟังชัน เพื่อตรวจสอบอินพุตพารามิเตอร์

Program Example 9.11: assert preconditions

```

1 # Assert preconditions
⇒2 def DIV(x, y):
⇒3     assert (y != 0), "Can't divide by zero!!!"
⇒4     return x / y
⇒5 print (DIV(5, 3))
⇒6 print (DIV(5, 0))

```



OUTPUT

```

1.6666666666666667
Traceback (most recent call last):
  File "C:/Python34/exam9_11.py", line 7, in <module>
    print (DIV(5, 0))
  File "C:/Python34/exam9_11.py", line 3, in DIV
    assert (y != 0), "Can't divide by zero!!!"
AssertionError: Can't divide by zero!!!

```

จากโปรแกรมที่ 9.11 เริ่มต้นบรรทัดที่ 2 โปรแกรมประกาศฟังชันชื่อ DIV() ทำหน้าที่คำนวณ การหารของเลข 2 จำนวน โดยมีพารามิเตอร์ 2 ตัวคือ x และ y บรรทัดที่ 3 โปรแกรมทำการตรวจสอบ สมมติฐานด้วยคำสั่ง assert ว่า y ต้องไม่เท่ากับ 0 จึงจะยอมให้โปรแกรมทำงานต่อไปได้ แต่ถ้า y เป็น 0 โปรแกรมจะพิมพ์ข้อความว่า “Can't divide by zero!!!” พร้อมกับแสดงข้อผิดพลาดทั้งหมดและยุติการทำงานของโปรแกรม สำหรับในกรณีที่ y ไม่เท่ากับ 0 โปรแกรมจะประมวลผลคำสั่งหาร x/y (บรรทัดที่ 4) และส่งผลลัพธ์จากการหารกลับไปยังผู้เรียกใช้งาน

บรรทัดที่ 5 โปรแกรมเรียกฟังชัน DIV พร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์เท่ากับ 5 และ 3 ตามลำดับ ผลลัพธ์ที่ได้จากฟังชัน DIV คือ 1.666 ซึ่งเป็นผลลัพธ์ที่ถูกต้องและไม่มีข้อผิดพลาด เนื่องจากค่าคงที่ 3 จะถูกกำหนดให้กับพารามิเตอร์ y และมีค่าไม่เท่ากับ 0

บรรทัดที่ 6 โปรแกรมเรียกฟังชัน DIV อีกรอบพร้อมกับค่าคงที่เป็นอาร์กิวเม้นต์เท่ากับ 5 และ 0 ตามลำดับ ผลลัพธ์ที่ได้จากฟังชัน DIV คือ ข้อความผิดพลาดที่ถูกจับได้โดย assert และข้อความผิดพลาดที่ไฟชอนสร้างขึ้นด้วย

ตัวอย่างโปรแกรมที่ 9.12 การตรวจสอบ postconditions ไว้หลังจากการเรียกฟังชัน เพื่อตรวจสอบเวลาพุ่มที่ส่งกลับ

Program Example 9.12: assert postconditions

```

1 # Assert postconditions
2 def DIV(x, y):
3     return x / y
⇒4 result = DIV(5, 6)
⇒5 assert result > 1, "Result less than zero"

```


OUTPUT

```
Traceback (most recent call last):
  File "C:/Python34/exam9_12.py", line 6, in <module>
    assert result > 1, "Result less than zero"
AssertionError: Result less than zero
```

จากตัวอย่างโปรแกรมที่ 9.12 ซึ่งเหมือนกับโปรแกรมที่ 9.11 แต่แตกต่างตรงที่ในตัวอย่างนี้ทำการตรวจสอบว่าหลังจากที่เรียกฟังชัน DIV แล้ว (บรรทัดที่ 5) ผลลัพธ์ที่ได้จากฟังชันจะถูกนำมาตรวจสอบในบรรทัดที่ 6 ว่าค่าที่ได้จากการคำนวณต้องเป็นค่าที่มากกว่า 1 เท่านั้น ถ้าเป็นเท็จ assert จะแสดงความผิดพลาดโดยพิมพ์ว่า “Result less than zero”

ตัวอย่างโปรแกรมที่ 9.13 เป็นการใช้ assert ในกรณีต่างๆ

Program Example 9.13: other assert

```
1 # Assert for any conditions
⇒2 x = 0
⇒3 assert x == 0, "X must be 0 only"
4
⇒5 f = lambda x: x*x
⇒6 assert f(2) == 4
⇒7 assert f(3) == 9
8
⇒9 y = 5
⇒10 value = 0
11 if y < 5:
12     value = -1
13 elif y > 5:
14     value = 1
15 else:
16     value = 0
⇒17 assert value == 0
```

จากโปรแกรมตัวอย่างที่ 9.13 แสดงการใช้งาน assert ในการตรวจสอบสมมติฐานในการนีต่างๆ โดยเริ่มจากบรรทัดที่ 3 เป็นการตรวจสอบว่าค่าในตัวแปร x ที่กำหนดไว้ (บรรทัดที่ 2) ยังคงเป็น 0 อยู่หรือไม่ เมื่อโปรแกรมทำงานต่อไปเรื่อยๆ แล้วสมมติว่ามีการเปลี่ยนแปลงค่าในตัวแปร x ให้มีค่าไม่เท่ากับ 0 โปรแกรมจะพิมพ์ข้อความว่า “X must be 0 only”

บรรทัดที่ 5 โปรแกรมทำการสร้างฟังชัน lambda ซึ่งทำหน้าที่ยกกำลังสองของ x โปรแกรมทำการตรวจสอบว่าด้านหน้าฟังชัน lambda (บรรทัดที่ 6 และ 7) เมื่อเรียกใช้ฟังชัน lambda พร้อมกับค่าคงที่เป็นอาเกิร์เมนต์เท่ากับ 2 ผลลัพธ์ที่ได้จะต้องเป็น 4 เท่านั้น (ถ้าไม่เท่ากับ 4 โปรแกรมจะแสดงข้อผิดพลาดจาก assert) เช่นเดียวกับบรรทัดที่ 7 เมื่อเรียกฟังชัน lambda พร้อมกับค่าคงที่เป็นอาเกิร์เมนต์เท่ากับ 3 ผลลัพธ์ที่ได้จะต้องเป็น 9 เท่านั้น

บรรทัดที่ 9 โปรแกรมทำการกำหนดค่าให้ตัวแปร y = 5 และ value = 0 เมื่อตรวจสอบเงื่อนในคำสั่ง if ในบรรทัดที่ 11 จะได้ผลลัพธ์เป็นเท็จ (เพราะ y มากกว่า 0) โปรแกรมจึงเลื่อนมาตรวจสอบเงื่อนไขต่อที่คำสั่ง elif ในบรรทัดที่ 13 จะได้ผลลัพธ์เป็นเท็จ (เพราะ y เท่ากับ 5) โปรแกรมจึงเลื่อนไป

ทำงานต่อที่คำสั่ง else ในบรรทัดที่ 15 โดยกำหนดค่าให้กับตัวแปร value = 0 เมื่อเสร็จจากคำสั่ง else แล้ว โปรแกรมจะตรวจสอบค่าใน value ด้วย assert โดยกำหนดไว้ว่าค่าในตัวแปร value ต้องเป็น 0 เท่านั้น ถ้า value เป็นค่าอื่นๆ ที่ไม่เท่ากับ 0 assert จะแจ้งความผิดพลาดทันที

ตัวอย่างโปรแกรมที่ 9.14 เป็นโปรแกรมแปลงค่าอุณหภูมิจาก Kelvin เป็น Fahrenheit

Program Example 9.14: Kelvin to Fahrenheit

```

1 # Convert Kelvin to Fahrenheit
2 def KelvinToFahrenheit(Temperature):
3     assert(Temperature >= 0), "Colder than absolute zero!"
4     return((Temperature - 273) * 1.8) + 32
5
6 print(KelvinToFahrenheit(273))
7 print(int(KelvinToFahrenheit(505.78)))
8 print(KelvinToFahrenheit(-5))

```



OUTPUT

```

32.0
451
Traceback (most recent call last):
  File "C:/Python34/exam9_14.py", line 8, in <module>
    print (KelvinToFahrenheit(-5))
  File "C:/Python34/exam9_14.py", line 3, in
    KelvinToFahrenheit
    assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!

```

จากโปรแกรมตัวอย่างที่ 9.14 เป็นการแปลงอุณหภูมิจาก Kelvin ไปเป็น Fahrenheit โดยโปรแกรมได้ทำการตั้งสมมติฐานว่าค่าของอุณหภูมิ (Temperature) จะต้องไม่เป็นค่าติดลบ (บรรทัดที่ 3) ถ้าค่าดังกล่าวที่รับเข้ามาประมวลผลในฟังชันเป็นค่าลบ assert จะแสดงความผิดพลาดออกมาเป็นข้อความคือ “Colder than absolute zero!”

บทที่ 10

การจัดการแฟ้มข้อมูล

(File Management)



1. แฟ้มข้อมูลคืออะไร?

แฟ้มข้อมูล (File) หมายถึง ข้อสนนเทศหรือข้อมูลที่มีขนาดเล็กที่สุดที่คอมพิวเตอร์สามารถแสดงให้ผู้ใช้เห็นได้ โดยจัดเก็บข้อมูลในลักษณะระเบียน (Record) แฟ้มจะถูกจัดเก็บอยู่ในหน่วยความจำภายนอก (External storage) ชนิดถาวร เช่น ฮาร์ดดิสก์ แ芬ชีดี เป็นต้น วัตถุประสงค์ในการจัดเก็บข้อมูลจะแตกต่างกันตามลักษณะการใช้งาน เช่น ข้อมูลที่ใช้ในสำนักงานส่วนใหญ่เป็นแฟ้มประเภทเวิร์ดโปรดเซสเซอร์ เช่น .doc, .xls, .xml เป็นต้น ข้อมูลเกี่ยวกับมัลติมีเดีย เช่น .wmv, .mp4, .dat หรือ .avi เป็นต้น ข้อมูลที่ใช้สำหรับประมวลผล เช่น .exe, .com เป็นต้น ซึ่งสามารถแบ่งตามชนิดของแฟ้มข้อมูลได้เป็น 2 ประเภทใหญ่ๆ คือ

- 1) แฟ้มข้อความ (Text file) สำหรับเก็บ ตัวอักษร ตัวเลข หรือข้อความ ที่สามารถอ่านทำความเข้าใจได้ ตัวอย่างแฟ้มข้อมูล เช่น .txt, .doc, .ini, .xml, .c, .py เป็นต้น
- 2) แฟ้มไบนาเรี่ย (Binary file) สำหรับใช้เก็บข้อมูลระบบ หรือคำสั่งของคอมพิวเตอร์ระดับล่างสุด และส่วนใหญ่ไม่สามารถอ่านทำความเข้าใจได้ ตัวอย่างแฟ้มข้อมูลประเภทนี้ เช่น .exe, .com, .msi, .mp3, .wmv, .dat, .zip, .class หรือ .o เป็นต้น

2. การบริหารจัดการกับแฟ้มข้อมูล (File Management)

การจัดการกับแฟ้มข้อมูลนั้นมีลำดับขั้นตอนการทำงาน 3 ขั้นตอน คือ

- 1) การเปิดแฟ้มข้อมูล มีคุณสมบัติ 4 ประการคือ
 - a. เปิดแฟ้มเพื่ออ่านอย่างเดียว ใช้สัญลักษณ์ r (read only)
 - b. เปิดแฟ้มเพื่อทำการเขียน ใช้สัญลักษณ์ w (write)
 - c. เปิดแฟ้มเพื่อทำการอ่านและเขียน ใช้สัญลักษณ์ rw (read-write)
 - d. เปิดแฟ้มเพื่อปรับปรุงข้อมูล ใช้สัญลักษณ์ a (append)
- 2) การใช้งานหรือดำเนินการกับแฟ้มข้อมูล ประกอบไปด้วย

- a. การอ่านและเขียนข้อมูล
 - b. การลบและแก้ไขข้อมูล
 - c. การค้นหาและแทนที่ข้อมูล
 - d. การบีบอัดและขยายแฟ้มข้อมูล
 - e. การแสดงและการกำหนดคุณสมบัติของแฟ้มข้อมูล (Permission)
 - f. และอื่นๆ
- 3) การปิดแฟ้มข้อมูล

การเปิดและปิดแฟ้ม (Opening and Closing Files)

การเปิดแฟ้ม (Opening files)

คำสั่งเปิดแฟ้มข้อมูลในไพธอนคือ คำสั่ง `open` ซึ่งเป็นชนิด built-in พังชัน คำสั่งจะทำการสร้างอオปเจกของแฟ้ม (File object) ขึ้นมา เพื่อใช้สำหรับอ้างอิงในการเรียกใช้งานแฟ้มต่อไป รูปแบบคำสั่งในการเปิดแฟ้มข้อมูลในไพธอน แสดงได้ดังนี้คือ

```
file object = open(file_name [, access_mode][, buffering])
```

- `object` คือ ออปเจกของแฟ้มที่ถูกใช้ในการอ้างอิงไปยังแฟ้มจริงๆ ที่เก็บอยู่ในหน่วยความจำชนิดถาวร เช่น อาร์ดดิสก์
- `open` คือ พังชันเปิดแฟ้มข้อมูล
- `file_name` คือ ชื่อของแฟ้มข้อมูลที่ผู้เขียนโปรแกรมต้องการใช้งาน
- `access_mode` คือ วิธีการเข้าใช้งานแฟ้ม เช่น อ่านอย่างเดียว เขียน หรือทั้งอ่านและเขียน เป็นต้น ซึ่งจะกำหนดด้วยตัวอักษร (โหมดในการเปิดแฟ้มข้อมูล) ดูได้จากตารางที่ 10.1 เมื่อไม่กำหนด `access_mode` ไพธอนจะถือว่าเป็นการเปิดแฟ้มเพื่ออ่านอย่างเดียว
- `buffering` คือ หน่วยความจำชั่วคราวเพื่อใช้สำหรับเก็บข้อมูลก่อนนำไปประมวลผล เนื่องจากการอ่านเขียนข้อมูลกับหน่วยความจำต้องทำงานได้ช้ามาก ดังนั้นไพธอนจำเป็นต้องสร้างแหล่งสำหรับพกข้อมูลเอาไว้ก่อน เมื่อข้อมูลเต็มจำนวนพื้นที่ที่กันไว้เป็นบัฟเฟอร์แล้ว โปรแกรมจึงค่อยนำเอาข้อมูลดังกล่าวไปประมวลผล ถ้ากำหนดค่า `buffer` เป็น 0 แสดงว่าไม่ใช้หน่วยความจำชนิดบัฟเฟอร์ ถ้ากำหนดเป็น 1 ไพธอนจะใช้บัฟเฟอร์ในการอ่านเขียนแฟ้มแบบที่ลับบรรทัด ถ้ากำหนดเป็นเลขจำนวนเต็มที่มากกว่า 1 คือ เป็นการจดหน่วยความจำตามจำนวนที่ระบุไว้ แต่ถ้าตัวเลขที่กำหนดเป็นค่าติด

ลบ “ไฟชอนจะใช้ค่าขนาดของบัฟเฟอร์เท่ากับจำนวนที่ระบบปฏิบัติการกำหนดไว้ (โดยปกติจะอยู่ที่ประมาณ 4 – 16 K สำหรับวินโดวส์)

ตารางที่ 10.1 แสดงอักษร หรือกลุ่มของตัวอักษรที่ใช้ในการเปิดแฟ้มข้อมูล (โหมด)

โหมด	คำอธิบาย
r	เปิดแฟ้มที่มีอยู่แล้วเพื่ออ่านเพียงอย่างเดียว ตัวชี้จะชี้ไปยังตำแหน่งเริ่มต้นของแฟ้ม (เป็นโหมดการใช้งานปกติ (default) ถ้าผู้เขียนโปรแกรมไม่กำหนดโหมดในการเปิดแฟ้มไว้)
rb	เปิดแฟ้มชนิดไบนารีเพื่ออ่านเพียงอย่างเดียว ตัวชี้อยู่ตำแหน่งเริ่มต้นของแฟ้ม
r+	เปิดแฟ้มเพื่ออ่านและเขียน โดยที่ข้อมูลเก่ายังคงอยู่ ตัวชี้อยู่ตำแหน่งเริ่มต้น
rb+	เปิดแฟ้มชนิดไบนารีเพื่ออ่านและเขียน โดยที่ข้อมูลเก่ายังคงอยู่ ตัวชี้อยู่ตำแหน่งเริ่มต้น
w	เปิดแฟ้มเพื่อเขียนเท่านั้น ในกรณีที่มีแฟ้มอยู่แล้ว ข้อมูลที่อยู่ในแฟ้มเดิมจะถูกเขียนทับ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่ ตัวชี้อยู่ตำแหน่งเริ่มต้น
wb	เปิดแฟ้มชนิดไบนารีเพื่อเขียนเท่านั้น ในกรณีที่มีแฟ้มอยู่แล้ว ข้อมูลที่อยู่ในแฟ้มเดิมจะถูกเขียนทับ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่ ตัวชี้อยู่ตำแหน่งเริ่มต้น
w+	เปิดแฟ้มเพื่ออ่านและเขียน ในกรณีที่มีแฟ้มอยู่แล้ว ข้อมูลที่อยู่ในแฟ้มเดิมจะถูกเขียนทับ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่ ตัวชี้อยู่ตำแหน่งเริ่มต้น
wb+	เปิดแฟ้มชนิดไบนารีเพื่ออ่านและเขียน ในกรณีที่มีแฟ้มอยู่แล้ว ข้อมูลที่อยู่ในแฟ้มเดิมจะถูกเขียนทับ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่ ตัวชี้อยู่ตำแหน่งเริ่มต้น
a	เปิดแฟ้มเพื่อเขียนข้อมูลต่อท้ายแฟ้ม ตัวชี้จะอยู่ในตำแหน่งสุดท้ายในแฟ้ม ในกรณีที่มีแฟ้มอยู่แล้วจะเขียนข้อมูลต่อ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่
ab	เปิดแฟ้มชนิดไบนารีเพื่อเขียนข้อมูลต่อท้ายแฟ้ม ตัวชี้จะอยู่ในตำแหน่งสุดท้ายในแฟ้ม ในกรณีที่มีแฟ้มอยู่แล้วจะเขียนข้อมูลต่อ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่
a+	เปิดแฟ้มเพื่ออ่านและเขียน ตัวชี้จะอยู่ตำแหน่งสุดท้ายของข้อมูลในแฟ้ม ในกรณีที่มีแฟ้มอยู่แล้วจะเขียนข้อมูลต่อ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่
ab+	เปิดแฟ้มชนิดไบนารีเพื่ออ่านและเขียน ตัวชี้จะอยู่ตำแหน่งสุดท้ายของข้อมูลในแฟ้ม ในกรณีที่มีแฟ้มอยู่แล้วจะเขียนข้อมูลต่อ ถ้าไม่มีแฟ้มข้อมูลอยู่ จะสร้างแฟ้มขึ้นใหม่
b	เปิดอ่านแฟ้มข้อมูลโดยไม่สนใจรหัสปิดท้าย (tg) ของแฟ้ม
U	เปิดอ่านแฟ้มข้อมูลประเภท Unicode ซึ่งมีความแตกต่างกันในเรื่องของรหัสปิดท้ายบรรทัด เช่น ถ้าเป็นแฟ้มทั่วๆ ไป จะใช้รหัสปิดท้ายบรรทัดเป็น \n แต่ถ้าเป็นแฟ้มบันวนิวโนว์ส์จะใช้ \nแทน เป็นต้น



Note: สัญลักษณ์ + ที่ใช้ในตารางที่ 10.1 คือ แฟ้มมีคุณสมบัติทั้งอ่านและเขียน ผู้เขียนโปรแกรมสามารถสมสัญลักษณ์ในการเปิดแฟ้มได้ เช่น r+ คือเปิดอ่านแฟ้ม Unicode เป็นต้น

คุณสมบัติที่เกี่ยวข้องกับแฟ้มข้อมูล

เมื่อแฟ้มข้อมูลถูกเปิดใช้งานแล้ว ผู้เขียนโปรแกรมสามารถตรวจสอบคุณสมบัติต่างๆ ของแฟ้มได้ดังนี้

- ตรวจสอบว่าแฟ้มข้อมูลถูกปิดหรือยัง มีรูปแบบคือ
file.closed พังชันส่งค่ากลับเป็นจริง เมื่อแฟ้มถูกปิดแล้ว
- ตรวจสอบว่าแฟ้มข้อมูลมีโหมดการเข้าใช้งานอย่างไร
file.mode พังชันส่งค่ากลับเป็นโหมดที่เปิดใช้งานอยู่
- ตรวจสอบชื่อของแฟ้มข้อมูล
file.name พังชันส่งค่ากลับเป็นชื่อของแฟ้ม

ตัวอย่างโปรแกรมที่ 10.1 แสดงคุณสมบัติของแฟ้มข้อมูลที่ถูกเปิดใช้งานอยู่

Program Example 10.1: file attributes

```

1 # Show information about file
2 # Open a file
3 f = open("MyFile.txt", "w")
4 print ("Name of the file: ", f.name)
5 print ("Closed or not : ", f.closed)
6 print ("Opening mode : ", f.mode)

```



Name of the file: MyFile.txt
Closed or not : False
Opening mode : w

จากโปรแกรมตัวอย่างที่ 10.1 เริ่มต้นบรรทัดที่ 3 โปรแกรมเปิดแฟ้มชื่อ MyFile.txt ด้วยคำสั่ง open ในโหมดเขียนอย่างเดียว (w) อ้อปเจ็กที่ได้จากคำสั่งเปิดแฟ้มเก็บไว้ในตัวแปร f เพื่อใช้อ้างอิงแฟ้มจริงในหน่วยความจำ บรรทัดที่ 4 โปรแกรมพิมพ์ชื่อแฟ้มโดยใช้คำสั่ง f.name ผลลัพธ์ที่ได้คือ MyFile.txt บรรทัดที่ 5 โปรแกรมตรวจสอบแฟ้มว่าถูกปิดหรือไม่ด้วยคำสั่ง f.closed ผลลัพธ์คือ False (แฟ้มยังไม่ถูกปิด) และบรรทัดที่ 6 โปรแกรมแสดงโหมดการใช้งานแฟ้มด้วยคำสั่ง f.mode ผลลัพธ์คือ w (เปิดแฟ้มเพื่อเขียน)

การปิดแฟ้ม (Closing files)

หลังจากผู้เขียนโปรแกรมใช้คำสั่งเปิดแฟ้มและดำเนินการใดๆ กับแฟ้มข้อมูลแล้ว จำเป็นต้องปิดแฟ้มทุกๆ ครั้ง ทั้งนี้เพราะข้อมูลต่างๆ ที่อยู่ในหน่วยความจำชั่วคราวอาจจะยังไม่ถูกเขียนกลับไปยัง

แฟ้มข้อมูลที่อยู่ในหน่วยความจำถาวร ดังนั้นคำสั่ง close (ปิดแฟ้ม) จะช่วยให้ข้อมูลเหล่านั้นถูกบันทึกในบางครั้งผู้เขียนโปรแกรมอาจจะลืมหรือโปรแกรมอาจเกิดข้อผิดพลาดขึ้น ทำให้แฟ้มที่ถูกเปิดไว้ไม่ได้รับการปิดลง ซึ่งส่งผลให้สิ่นเปลืองเนื่องที่ของหน่วยความจำ ไฟชอนจึงได้เตรียมมาตรการที่ช่วยจัดการกับปัญหาเหล่านี้ให้ระดับหนึ่ง คือ เมื่อผู้ใช้งานเปิดแฟ้มไว้แต่ไม่ได้สั่งให้โปรแกรมทำการปิดแฟ้มไฟชอนจะปิดแฟ้มดังกล่าวให้เองโดยอัตโนมัติ เมื่อผู้ใช้งานได้ทำการเปิดแฟ้มอีกครั้งขึ้นใหม่ รูปแบบคำสั่งในการปิดแฟ้มข้อมูลในไฟชอนแสดงได้ดังนี้คือ

`fileObject.close()`

`fileObject` คือ อ็อปเจกของแฟ้มที่ต้องการปิด และ `close` คือ เมธอดที่ใช้ในการปิดแฟ้มข้อมูล

สำหรับตัวอย่างปิดแฟ้มดังโปรแกรมที่ 10.2

ตัวอย่างโปรแกรมที่ 10.2

Program Example 10.2: `close file`

```

1   # Show Opening and Closing files
2   FilePath = "C:\\Python34\\ทดสอบอ่านไทย.txt"
3   try:
4       f = open(FilePath, "rU")
5       if f:
6           print("สามารถเปิดแฟ้ม :",FilePath, "ได้แล้ว")
7           print ("Name of the file: ", f.name)
8           print ("Opening mode : ", f.mode)
9           str = f.readline()
10          print("ข้อความในแฟ้มคือ ",str)
11      except IOError as err:
12          print("ไม่สามารถเปิดแฟ้มได้ เพราะ : ",err.args)
13      else:
14          print ("ปิดแฟ้มเรียบร้อยแล้ว")
15          f.close()

```

แฟ้มข้อมูลอินพุตที่ใช้ทดสอบ



Input File: ทดสอบอ่านไทย.txt

แฟ้มนี้ใช้สำหรับทดสอบการเปิดและการปิดแฟ้ม

ผลลัพธ์ในการนี้ที่โปรแกรมเปิดแฟ้มข้อมูลเพื่ออ่านได้



OUTPUT

สามารถเปิดแฟ้ม : C:\\Python34\\ทดสอบอ่านไทย.txt ได้แล้ว

Name of the file: C:\\Python34\\ทดสอบอ่านไทย.txt

Opening mode : rU

ปิดแฟ้มเรียบร้อยแล้ว

ผลลัพธ์ในการนี้ที่โปรแกรมไม่สามารถเปิดแฟ้มข้อมูลเพื่ออ่านได้ เช่น ไม่มีแฟ้มอยู่ในไดเรคทรอรี่



OUTPUT

ไม่สามารถเปิดแฟ้มได้ เพราะ :(2, 'No such file or directory')

จากโปรแกรมตัวอย่างที่ 10.2 บรรทัดที่ 2 เป็นการกำหนดที่อยู่ของแฟ้มที่ต้องการใช้งานชื่อว่า "ทดสอบอ่านไทย.txt" ให้กับตัวแปร filePath ภายในแฟ้มดังกล่าวมีข้อความว่า "แฟ้มนี้ใช้สำหรับทดสอบการเปิดและการปิดแฟ้ม" ลำดับถัดไปบรรทัดที่ 3 โปรแกรมดักจับความผิดพลาดที่อาจจะเกิดขึ้นจากการเปิดแฟ้มข้อมูล บรรทัดที่ 4 โปรแกรมเปิดแฟ้มข้อมูลชนิด Unicode (แฟ้มภาษาไทย) ในโหมดอ่านได้อย่างเดียว ("rb") ถ้าโปรแกรมเปิดแฟ้มไม่สำเร็จจะไปทำงานหลังคำสั่ง except (บรรทัดที่ 11) ซึ่งจะพิมพ์ข้อความรายงานความผิดพลาดที่เกิดขึ้นออกทางจอภาพคือ "ไม่สามารถเปิดแฟ้มได้ เพราะ" พร้อมกับความผิดพลาดที่เกิดขึ้นในตัวแปร err.args

แต่ถ้าแฟ้มสามารถเปิดใช้งานได้ตามปกติ โปรแกรมจะทำงานที่คำสั่ง if (บรรทัดที่ 5) ซึ่งตรวจสอบเงื่อนไขว่าอ้อปเจ็ก f มีค่าจริงหรือไม่ (ถ้า `f == None` หรือ 0 แสดงว่าไม่สามารถเปิดแฟ้มได้) ถ้าเงื่อนไขเป็นจริง โปรแกรมจะทำงานในบรรทัดที่ 6 คือสั่งพิมพ์ข้อความว่า "สามารถเปิดแฟ้ม : C:\Python34\ทดสอบอ่านไทย.txt ได้แล้ว" บรรทัดที่ 7 พิมพ์ชื่อแฟ้มเท่ากับ "C:\Python34\ทดสอบอ่านไทย.txt" บรรทัดที่ 8 พิมพ์โหมดการอ่านแฟ้มเท่ากับ "rb" บรรทัดที่ 9 โปรแกรมทำการอ่านข้อมูลจากแฟ้มมา 1 บรรทัดด้วยคำสั่ง `readline` และพิมพ์ข้อความที่อ่านได้ออกจอภาพ เมื่อโปรแกรมทำงานดังกล่าวเสร็จแล้ว จะไปทำงานที่บรรทัดที่ 13 หลังคำสั่ง else โดยพิมพ์ข้อความว่า "ปิดแฟ้มเรียบร้อยแล้ว" พร้อมกับปิดแฟ้มข้อมูล



Note: การระบุเส้นทาง (path) เพื่อเปิดแฟ้มในวินโดวส์ให้ใช้เครื่องหมาย \ ในการระบุที่อยู่ เช่น `filePath = "C:\Python34\ทดสอบอ่านไทย.txt"`

การดำเนินการกับแฟ้มข้อมูล (File Operating)

การอ่านแฟ้มข้อมูล

การเขียนโปรแกรมเพื่ออ่านข้อมูลของแฟ้มอุปกรณ์แสดงผลนิยมใช้เมธอด `read()`, `readline()` และ `readlines` ซึ่งมีรูปแบบคำสั่งดังนี้

`f.read([size])`

โดย `f` คือ อ้อปเจ็กของแฟ้มข้อมูลจริงที่ต้องการอ้างถึง, `read` คือ คำสั่งในการอ่านแฟ้มข้อมูล และ `[size]` คือ จำนวนตัวอักษรที่ต้องการอ่านจากแฟ้ม ถ้ากำหนดค่า `size` เป็นค่าลบหรือไม่ได้ระบุไว้ ไฟรอนจะตีความว่าอ่านข้อมูลทั้งแฟ้ม

`f.readline([size])`

`readline` อ่านข้อมูลจากแฟ้มครั้งละ 1 บรรทัดโดยอัตโนมัติ แต่ผู้เขียนโปรแกรมสามารถกำหนดจำนวนของตัวอักษรที่ต้องการอ่านได้ โดยกำหนดใน `size` คำสั่ง `readline` จะอ่านแฟ้มแต่ละบรรทัดโดย

พิจารณาการหัසการขีนบรรทัดใหม่ \t โปรแกรมจะหยุดอ่านก็ต่อเมื่อพบรหัสจบแฟ้ม EOF (end of file)

f.readlines([sizehint])

readlines อ่านข้อมูลจากบรรทัดแรกจนจบแฟ้ม ข้อมูลที่ส่งกลับประกอบไปด้วยข้อมูลหลาย ๆ บรรทัด ถ้ากำหนดค่าใน sizehint คำสั่ง readline จะเปลี่ยนจากการอ่านแฟ้มทั้งหมดเป็นการอ่านตามจำนวนที่ระบุใน sizehint ซึ่งค่าที่กำหนดใน sizehint จะมีหน่วยเป็นไบต์ (1 ไบต์เท่ากับ 8 บิต)



Note: เมื่อแฟ้มไม่มีข้อมูลใดๆ (แฟ้มว่าง) คำสั่ง read จะส่งกลับค่า null กลับไปให้ผู้เรียกใช้ หรือมีความยาวเท่ากับ 0 แต่ถ้าเป็นบรรทัดว่างความยาวของบรรทัดว่างจะมีค่าเท่ากับ 1

ตัวอย่างโปรแกรมที่ 10.3 แสดงการใช้คำสั่ง read, readline และ readlines

Program Example 10.3: *read, readline, readlines*

```

1   # Testing read, readline(s) method
2   FilePath = "C:\Python34\README.txt"
3
4   try:
5       f = open(FilePath)
6       #Testing read method
7       str = f.read()
8       print(str)
9       #Testing readlines method
10      f = open(FilePath)
11      str = f.readlines(15)
12      print(str)
13      #Testing readline method
14      f = open(FilePath)
15      while 1:
16          line = f.readline()
17          if len(line):
18              print(line)
19          else: break
20      except IOError as err:
21          print("Can't open file because : ",err.args)
22      else:
23          print("This file was closed!")
f.close()
```

ผลลัพธ์ของการทดสอบคำสั่ง read



OUTPUT

This is Python version 3.4.0 beta 1

=====

Copyright (c) 2001, 2002, 2003, 2004, 2005,...

.....

All trademarks referenced herein are property of their respective holders.

This file was closed!
ผลลัพธ์ของการทดสอบคำสั่ง `readlines`

 OUTPUT
['This is Python version 3.4.0 beta 1\\n']
This file was closed!

ผลลัพธ์ของการทดสอบคำสั่ง `readline`

 OUTPUT
This is Python version 3.4.0 beta 1
=====Copyright (c) 2001, 2002, 2003, 2004, 2005,...
.....All trademarks referenced herein are property of their respective holders.
This file was closed!

จากตัวอย่างโปรแกรมที่ 10.3 แสดงตัวอย่างการใช้งานคำสั่ง `read`, `readline` และ `readlines` เริ่มต้นในบรรทัดที่ 2 โปรแกรมกำหนดที่อยู่ของแฟ้มชื่อ `README.txt` ให้กับตัวแปร `FilePath` ต่อจากนั้นบรรทัดที่ 4 โปรแกรมจะทำการเปิดแฟ้มด้วยคำสั่ง `open` ซึ่งเป็นการเปิดแฟ้มเพื่ออ่านเท่านั้น อ้อปเจ็กที่ได้จากการเปิดแฟ้มจะถูกเก็บไว้ใน `f` เพื่อใช้สำหรับอ้างอิงแฟ้ม บรรทัดที่ 6 โปรแกรมอ่านข้อมูลจากแฟ้มในครั้งเดียวด้วยคำสั่ง `read` และพิมพ์ข้อความทั้งหมดออกจากทางจอภาพ (บรรทัดที่ 7)

ลำดับถัดไปบรรทัดที่ 9 โปรแกรมทำการเปิดแฟ้มใหม่ด้วยคำสั่ง `open` (เนื่องจากการอ่านแฟ้มด้วยคำสั่ง `read` ในครั้งแรกทำให้ตัวชี้ตำแหน่งของแฟ้มชี้ไปยังตำแหน่งท้ายแฟ้มแล้ว เมื่อเปิดแฟ้มใหม่ ตัวชี้จะเริ่มที่ต้นแฟ้ม) บรรทัดที่ 10 โปรแกรมทำการอ่านแฟ้มด้วยคำสั่ง `readlines` ในที่นี้ได้กำหนด `sizehint` ให้มีค่าเท่ากับ 15 นั่นคือ แฟ้มจะถูกอ่านข้อมูลเข้ามา 1 บรรทัด (ถ้าสมมติว่า 1 บรรทัดมี 30 ตัวอักษร เมื่อกำหนดค่า `sizehint` มีค่าระหว่าง 1 - 30 ข้อมูลที่อ่านได้จะเท่ากับ 1 บรรทัด) และพิมพ์ข้อความทั้งหมดออกจากทางจอภาพ (บรรทัดที่ 11)

ลำดับถัดไปบรรทัดที่ 13 โปรแกรมเปิดแฟ้มใหม่เพื่อให้ตัวชี้ชี้ที่ตำแหน่งเริ่มต้นแฟ้ม บรรทัดที่ 14 โปรแกรมทำการอ่านข้อมูลที่ละบรรทัดไปเรื่อยๆ จนกว่าจะหมดแฟ้มด้วยคำสั่ง `while 1` บรรทัดที่ 15 โปรแกรมอ่านข้อมูลแบบทีละบรรทัดโดยใช้คำสั่ง `readline` ข้อความที่อ่านได้นำไปเปรียบเทียบด้วยคำสั่ง `if` (บรรทัดที่ 16) ว่าโปรแกรมอ่านแฟ้มข้อมูลหมดหรือยัง (ถ้าความยาวของข้อมูลที่อ่านได้มีค่าเท่ากับ 0 แสดงว่าข้อมูลหมดแฟ้มแล้ว แต่ถ้าความยาวเท่ากับ 1 แสดงว่าเป็นบรรทัดว่าง) เมื่อความยาวไม่เป็น 0 โปรแกรมทำการพิมพ์ข้อความออกจากทาง (บรรทัดที่ 17) แต่ถ้าความยาวเป็น 0 โปรแกรมจะหยุดการทำงานของ `while` ด้วยคำสั่ง `break` (บรรทัดที่ 18) ผลจากคำสั่ง `break` ทำให้โปรแกรมทำงานหลังคำสั่ง `else` คือการปิดแฟ้มข้อมูล (บรรทัดที่ 23) แต่ถ้าโปรแกรมไม่สามารถเปิดแฟ้มได้จะเกิดข้อผิดพลาดขึ้น โดยโปรแกรมจะทำงานหลังคำสั่ง `except IOError` (บรรทัดที่ 19)

การอ่านแฟ้มข้อมูลที่ลับบรรทัด

ในบางกรณีผู้เขียนโปรแกรมต้องการอ่านแฟ้มข้อมูลเข้ามาประมาณผลที่ลับบรรทัด โดยไม่จำเป็นต้องเปิดแฟ้มข้อมูลด้วยคำสั่ง open ซึ่งไฟรอนได้จัดเตรียมเมธอดไว้ให้คือ getline ซึ่งอยู่ในโมดูล linecache มีรูปแบบคำสั่งใช้งานดังนี้

```
linecache.getline(file_name, line_no)
```

โดย linecache คือ ชื่อโมดูล, getline คือชื่อของเมธอดที่ทำหน้าที่อ่านแฟ้มข้อมูลเข้ามาที่ลับบรรทัด, file_name คือ ชื่อแฟ้มที่ต้องการเปิดใช้งาน และ line_no คือ จำนวนบรรทัดที่ต้องการอ่าน เมื่ออ่านข้อมูลแล้วไม่จำเป็นต้องปิดแฟ้ม แต่ควรคืนหน่วยความจำให้กับระบบด้วยเมธอด clearcache() สำหรับรูปแบบการใช้งานแสดงในโปรแกรมที่ 10.4

ตัวอย่างโปรแกรมที่ 10.4 แสดงการใช้คำสั่ง getline

Program Example 10.4: linecache.getline

```
1 # read file in line by line
⇒2 import linecache
⇒3 FilePath = "C:\Python34\README.txt"
⇒4 for line in range(5):
⇒5     print(linecache.getline(FilePath, line))
⇒6 linecache.clearcache()
```



OUTPUT

```
This is Python version 3.4.0 beta 1
=====
Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007,
2008, 2009, 2010, 2011,
```

จากตัวอย่างโปรแกรมที่ 10.4 โปรแกรมเริ่มต้นในบรรทัดที่ 2 โดยการ import โมดูล linecache เข้ามาทำงานในโปรแกรม จากนั้นบรรทัดที่ 3 โปรแกรมกำหนดตำแหน่งที่อยู่ของแฟ้มที่ต้องการเรียกใช้งานชื่อ README.txt ขึ้นตอนต่อไปโปรแกรมใช้คำสั่ง for (บรรทัดที่ 4) เพื่ออ่านอ่านแฟ้มจำนวน 5 บรรทัดเข้ามาทำงานในโปรแกรม บรรทัดที่ 5 โปรแกรมเรียกใช้เมธอด getline() เพื่ออ่านแฟ้มข้อมูลที่ลับบรรทัดโดยมีพารามิเตอร์ 2 ตัวคือ ชื่อแฟ้ม (FilePath) และตำแหน่งบรรทัดที่ต้องการอ่าน (line) โปรแกรมจะอ่านแฟ้มที่ลับ 1 บรรทัดจำนวน 5 บรรทัดพร้อมกับพิมพ์ข้อมูลในแต่ละบรรทัดออกจากภาพ เมื่อเสร็จสิ้นการอ่านแฟ้มแล้วโปรแกรมจะทำการเคลียร์ค่าข้อมูลที่อยู่ในหน่วยความจำโดยใช้เมธอด clearcache (บรรทัดที่ 6)

การอ่านแฟ้มข้อมูลที่ลับคำ

ในบางกรณีผู้เขียนโปรแกรมต้องการอ่านข้อมูลจากแฟ้มทีละคำ ผู้เขียนโปรแกรมสามารถใช้เมธอด `split()` ช่วยในการแยกคำออกจากแฟ้ม ซึ่งแสดงตัวอย่างการใช้งานดังในโปรแกรมที่ 10.5

ตัวอย่างโปรแกรมที่ 10.5 แสดงการใช้คำสั่ง `split`

Program Example 10.5: `split`

```

1 # read file in word by word
2 FilePath = "C:\Python34\README.txt"
3 wordTemp = []
4 wordCount = 0
5 file = open(FilePath, 'r')
6 for line in file:
7     for word in line.split():
8         wordTemp.append(word)
9     wordCount = wordCount + 1
10 print(wordTemp)
11 print(wordCount)

```



OUTPUT

```

['This', 'is', 'Python', 'version', '3.4.0', 'beta', '2',
'=====', 'Copyright', '(c)',
'2001,', '2002,', '2003,', '2004,',

...
code', 'but', 'these', 'are', 'entirely', 'optional.', 'All',
'trademarks', 'referenced', 'herein', 'are', 'property',
'of', 'their', 'respective', 'holders.']

```

The number of word count in this file = 973

จากโปรแกรมที่ 10.5 เป็นตัวอย่างการอ่านแฟ้มแบบคำต่อคำและนำมาเก็บไว้ในตัวแปรชนิดลิสต์ โดยโปรแกรมเริ่มต้นในบรรทัดที่ 2 เป็นการกำหนดชื่อแฟ้มข้อมูลที่ต้องการอ่านชื่อ README.txt บรรทัดที่ 3 กำหนดค่าตัวแปร `wordTemp` เป็นลิสต์ว่าง (`[]`) เพื่อเก็บคำที่อ่านได้จากแฟ้ม และบรรทัดที่ 4 กำหนดค่าตัวแปร `wordCount` เท่ากับ 0 เพื่อใช้ในการนับคำที่อ่านได้จากแฟ้ม ขั้นตอนต่อไปบรรทัดที่ 5 โปรแกรมเปิดแฟ้มชื่อ README.txt เพื่ออ่านอย่างเดียว เมื่อเปิดแฟ้มเสร็จแล้วโดยไม่มีข้อผิดพลาด ได้ๆ ก็ได้ขึ้น โปรแกรมจะใช้คำสั่ง `for` เพื่ออ่านข้อมูลจากแฟ้มเข้ามาทีละบรรทัด (บรรทัดที่ 6) ข้อมูลที่อ่านได้ในแต่ละบรรทัดจะถูกเก็บไว้ในตัวแปรชื่อ `line` จากนั้นบรรทัดที่ 7 โปรแกรมเรียกใช้เมธอดชื่อ `split` เพื่อแยกข้อมูลในตัวแปร `line` ออกเป็นคำๆ กัน นำไปเก็บในลักษณะแบบต่อคำโดยใช้คำสั่ง `append` (บรรทัดที่ 8) เก็บไว้ในตัวแปร `wordTemp` พร้อมกับนับจำนวนคำที่อ่านเข้ามาเก็บไว้ในตัวแปร `wordCount` (บรรทัดที่ 9) เมื่อโปรแกรมอ่านแฟ้มข้อมูลหมดแล้วจะทำการพิมพ์คำทั้งหมดที่เก็บไว้ในตัวแปร `wordTemp` และออกทางจอภาพ (บรรทัดที่ 10) คำสั่งสุดท้ายจะพิมพ์จำนวนคำที่นับได้ทั้งหมด ออกจากภาพ (บรรทัดที่ 11)

ตัวชี้ตำแหน่งในแฟ้มข้อมูล

การเขียนโปรแกรมกับแฟ้มข้อมูลบางครั้งผู้เขียนมีความจำเป็นต้องการหาตำแหน่งของตัวชี้ในแฟ้มที่กำลังทำงานอยู่ในขณะนั้น หรือต้องการอ่าน-เขียนแฟ้มในตำแหน่งที่ผู้เขียนโปรแกรมต้องการระบุเอง โดยปกติจะมีการใช้คำสั่ง `seek` สำหรับการย้ายตำแหน่งตัวชี้ไปยังตำแหน่งต่างๆ ในแฟ้มข้อมูล แต่ก็มีวิธีการอื่นๆ เช่น `tell` ที่ใช้สำหรับทราบตำแหน่งปัจจุบันของตัวชี้ หรือ `read` ที่ใช้สำหรับอ่านข้อมูลในแฟ้ม

`seek(offset[, from])`

โดย `seek` คือ คำสั่งที่ใช้เคลื่อนย้ายตำแหน่งตัวชี้ไปยังตำแหน่งต่างๆ ในแฟ้มข้อมูล `offset` คือ จำนวนข้อมูลที่ต้องการเคลื่อนย้ายไป (มีหน่วยเป็นไบต์) และ `from` ใช้ระบุตำแหน่งอ้างอิงเพื่อ เคลื่อนย้ายไปยังตำแหน่งต่างๆ ในแฟ้ม ถ้ากำหนด `from` มีค่าเท่ากับ 0 หมายถึงคำสั่ง `seek` จะอ้างอิง จากจุดเริ่มต้นของแฟ้ม ถ้ากำหนดค่าเท่ากับ 1 หมายถึง ใช้ตำแหน่งปัจจุบันในการอ้างอิง และกำหนดค่า เท่ากับ 2 หมายถึง อ้างอิงจากด้านท้ายของแฟ้ม ดังตัวอย่างโปรแกรมที่ 10.6



Input File: TEST.txt
This is Python version 3.4.0

ตัวอย่างโปรแกรมที่ 10.6 แสดงการใช้คำสั่ง `tell` และ `seek`

Program Example 10.6: tell and seek

```

1   # tell and seek method
2   file = open("TEST.txt", "r+")
3   str = file.read(10);
4   print ("Read String is : ", str)
5   # Check current position
6   position = file.tell();
7   print ("1.Current file position after read 1:", position)
8   # Reposition pointer at the beginning once again
9   position = file.seek(5, 0);
10  print ("2.Current file position after seek 1:", position)
11  str = file.read(10);
12  print ("Again read String is : ", str)
13  print ("2.Current file position after read 2:", file.tell())
14  # Reposition pointer at the current position
15  position = file.seek(0, 1);
16  print ("3.Current file position after seek 2:", position)
17  str = file.read(10);
18  print ("3.Again read String is : ", str)
19  print ("3.Current file position after read 3: ", position)
20  # Reposition pointer at the end of file
21  position = file.seek(0, 2);
22  print ("4.Current file position after seek 3:", position)
23  str = file.read(10);

```

```

⇒24 print ("4.Again read String is : ", str)
⇒25 print ("4.Current file position after read 4: ", position)
26   # Close open file
⇒27 file.close()

```



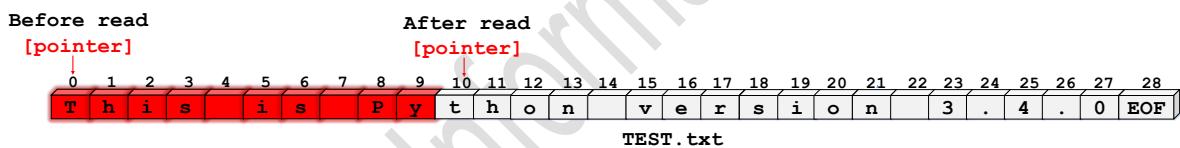
OUTPUT

```

1.Read String is : This is Py
1.Current file position after read 1: 10
2.Current file position after seek 1: 5
2.Again read String is : is Python
2.Current file position after read 2: 15
3.Current file position after seek 2: 15
3.Again read String is : version 3.
3.Current file position after read 3: 25
4.Current file position after seek 3: 28
4.Again read String is :
4.Current file position after read 4: 28

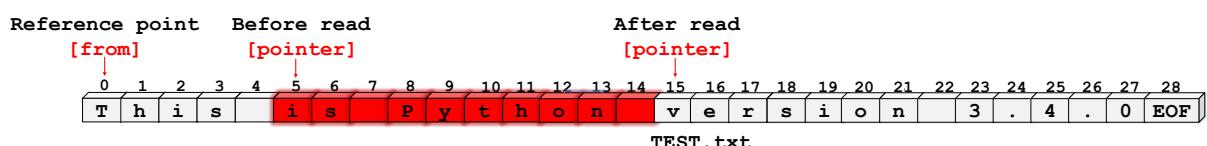
```

จากโปรแกรมที่ 10.6 แสดงการใช้เมธอด `tell` และ `seek` บรรทัดที่ 2 โปรแกรมเริ่มต้นด้วยการเปิดแฟ้มชื่อ `TEST.txt` เพื่ออ่านและเขียนโดยที่ข้อมูลเดิมยังคงอยู่ ในแฟ้มดังกล่าวมีข้อความคือ “`This is Python version 3.4.0`” เก็บอยู่ ขั้นตอนต่อไปบรรทัดที่ 3 โปรแกรมทำการอ่านแฟ้มจำนวน 10 ตัวอักษร เก็บไว้ในตัวแปร `str` เมื่อพิมพ์ข้อมูลจะได้ผลลัพธ์คือ “`This is Py`” (ในบรรทัดที่ 4) สำหรับตำแหน่งของตัวชี้ป้าจุบันจะอยู่ที่ตำแหน่ง 10 (อักษรตัว t) แสดงในรูปที่ 10.1



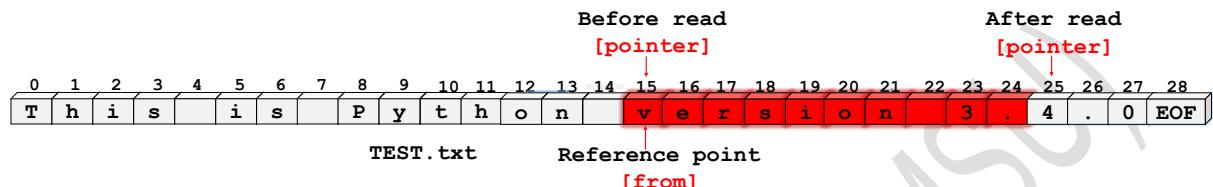
รูปที่ 10.1 แสดงการอ่านแฟ้มด้วยเมธอด `read(10)`

ลำดับต่อไปโปรแกรมต้องการแสดงตำแหน่งของตัวชี้แฟ้มป้าจุบัน โดยใช้เมธอด `tell` (บรรทัดที่ 6) ผลลัพธ์ของตำแหน่งตัวชี้ป้าจุบันเก็บไว้ในตัวแปร `position` เมื่อพิมพ์ค่าในตัวแปรดังกล่าว (บรรทัดที่ 7) ออกมายจะได้ผลลัพธ์เท่ากับ 10 (ตำแหน่งของตัวอักษร t) ต่อจากนั้นบรรทัดที่ 9 โปรแกรมใช้คำสั่ง `seek(5, 0)` เพื่อเลื่อนตำแหน่งตัวชี้กลับไปยังตำแหน่งเริ่มต้นของแฟ้ม (`from = 0`) และนับอักษรจากต้นแฟ้มไปอีก 5 (`offset = 5`) ตัวอักษร ดังนั้นทำให้ตัวชี้ป้าจุบันจะอยู่ตำแหน่งที่ 5 (ตัวอักษร i และนับอักษรจากต้นแฟ้มไปอีก 5 (`offset = 5`) ตัวอักษร ดังนั้นทำให้ตัวชี้ป้าจุบันจะอยู่ตำแหน่งที่ 5 (ตัวอักษร i แสดงในรูปที่ 10.2) จากนั้นบรรทัดที่ 11 โปรแกรมออกคำสั่งให้อ่านแฟ้มข้อมูลไปอีก 10 ตัวอักษรเก็บไว้ใน `str` เมื่อพิมพ์ข้อมูลที่อยู่ใน `str` จะได้ผลลัพธ์คือ “`is Python`” สำหรับตำแหน่งตัวชี้ป้าจุบันหลังจากอ่านแฟ้มแล้วจะอยู่ตำแหน่งที่ 15 แสดงดังรูปที่ 10.2



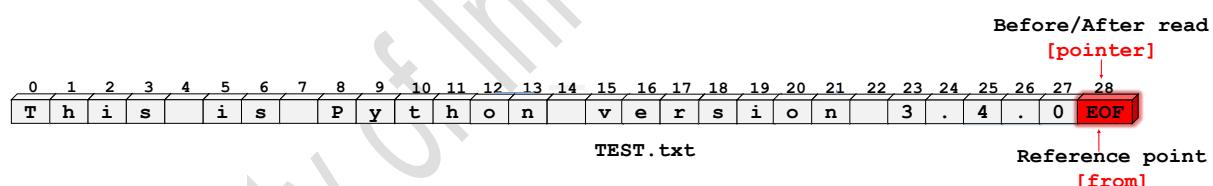
รูปที่ 10.2 แสดงการเลื่อนตำแหน่งตัวชี้ด้วยเมธอด `seek(5, 0)` และอ่านแฟ้มด้วย `read(10)`

ลำดับถัดไปบรรทัดที่ 15 โปรแกรมเลื่อนตำแหน่งตัวชี้โดยอ้างอิงจากตำแหน่งที่อยู่ปัจจุบัน (from = 1) และค่าของ offset จะต้องถูกกำหนดให้มีค่าเท่ากับ 0 เท่านั้น (มิฉะนั้นโปรแกรมจะเกิดผิดพลาด) เมื่อโปรแกรมพิมพ์ค่าตัวชี้ (บรรทัดที่ 16) หลังจากใช้คำสั่ง seek(0, 1) ตำแหน่งตัวชี้มีค่าเท่ากับ 15 จากนั้นบรรทัดที่ 17 โปรแกรมเริ่มอ่านตัวอักษรไปอีก 10 ตัวเริ่มจากตำแหน่งตัวอักษรตัวที่ 15 (อักษร v) เก็บไว้ในตัวแปร str เมื่อพิมพ์ค่าในตัวแปรดังกล่าว (บรรทัดที่ 18) ผลลัพธ์ที่ได้คือ "version 3." และเมื่อพิมพ์ค่าตำแหน่งตัวชี้ปัจจุบัน (บรรทัดที่ 19) จะมีค่าเท่ากับ 25 แสดงในรูปที่ 10.3



รูปที่ 10.3 แสดงการเลื่อนตำแหน่งตัวชี้ด้วยเมธอด seek(0, 1) และอ่านแฟ้มด้วย read(10)

ในลำดับถัดไปบรรทัดที่ 21 โปรแกรมทำการเลื่อนตัวชี้ไปยังตำแหน่งท้ายแฟ้มด้วยคำสั่ง seek(0, 2) โดยกำหนดพารามิเตอร์ from เท่ากับ 2 และค่า offset ต้องเท่ากับ 0 เท่านั้น ทำให้ตัวชี้ปัจจุบันอยู่ที่ตำแหน่งที่ 28 ซึ่งเป็นตำแหน่งสุดท้ายของแฟ้มข้อมูล ต่อจากนั้นบรรทัดที่ 23 โปรแกรมทำการอ่านตัวอักษรอีก 10 ตัว โดยเริ่มอ่านจากตำแหน่งที่ 28 (EOF) ซึ่งในกรณีนี้จะไม่มีตัวอักษรให้อ่านแล้ว เพราะตัวชี้ตำแหน่งอยู่ที่ท้ายแฟ้ม ผลลัพธ์ที่ได้จะพิมพ์ค่าว่างออกจอภาพ (บรรทัดที่ 24) และตำแหน่งตัวชี้ปัจจุบันจะอยู่ตำแหน่งที่ 28 (บรรทัดที่ 25) ดังรูปที่ 10.4



รูปที่ 10.4 แสดงการเลื่อนตำแหน่งตัวชี้ด้วยเมธอด seek(0, 2) และอ่านแฟ้มด้วย read(10)

การเขียนเพิ่มข้อมูล

การเขียนหรือการปรับปรุงข้อมูลในแฟ้มสำหรับภาษาไทยอนนิยมใช้เมธอด write() และ writelines() สำหรับคำสั่ง print>> ไฟรอน 3 ได้ยกเลิกไปแล้ว มีรูปแบบคำสั่งดังนี้

`f.write(string)`

โดย `f` คือ อ็อปเจ็กของแฟ้มข้อมูลจริงที่ต้องการอ้างถึง, `write` คือ คำสั่งในการเขียนแฟ้มข้อมูล และ `string` คือ ข้อความที่ต้องการเขียนลงแฟ้ม

f.**writelines**(sequence)

โดย f คือ อ็อปเจ็กของแฟ้มข้อมูลจริงที่ต้องการอ้างถึง, writelines คือ คำสั่งในการเขียนแฟ้มข้อมูล และ sequence คือ ชุดของข้อความที่ต้องการเขียนลงแฟ้ม สำหรับตัวอย่างการเขียนแฟ้มข้อมูลแสดงดังตัวอย่างโปรแกรมที่ 10.7

ตัวอย่างโปรแกรมที่ 10.7 แสดงการใช้คำสั่ง write และ writelines

Program Example 10.7: write vs writelines

```

1 # write, writelines
⇒2 string1 = "hello world in the new file\n"
⇒3 string2 = "and another line\n"
⇒4 sequence = ["Sequence string line 1\n", "Sequence string line 2"]
⇒5 file = open("newfile.txt", "w")
⇒6 file.write(string1)
⇒7 file.write(string2)
⇒8 file.writelines(sequence)
⇒9 file.close()

```



OUTPUT: myfile.txt

OUTPUT

```

hello world in the new file
and another line
Sequence string line 1
Sequence string line 2

```

จากตัวอย่างโปรแกรมที่ 10.7 บรรทัดที่ 2 โปรแกรมทำการกำหนดข้อความให้กับ string1

เท่ากับ “hello world in the new file\n”, string2 เท่ากับ “and another line\n” (บรรทัดที่ 2) และรายการของข้อความไว้ใน sequence เท่ากับ [“Sequence string line 1\n”, “Sequence string line 2”] (บรรทัดที่ 3) ลำดับถัดไปบรรทัดที่ 5 โปรแกรมทำการเปิดแฟ้มชื่อ myfile.txt ในโหมดเขียนทับ (w) และสั่งให้ทำการเขียนแฟ้มด้วยคำสั่ง write (บรรทัดที่ 6) กับ string1 และ string2 (บรรทัดที่ 7) สำหรับคำสั่ง writelines จะใช้เขียนข้อมูลกับค่าที่อยู่ตัวแปร sequence (บรรทัดที่ 8)

เมื่อเปิดแฟ้มชื่อ myfile.txt ในไดเรกทรอรีปัจจุบันจะปรากฏข้อความดังแสดงในตัวอย่างເອົາໆພຸດ
ຂ້າງບນ

การใช้คำสั่ง with กับแฟ้มข้อมูล

ไฟลอนໄດ້ເສັນຄວາມສັ່ງທີ່ໃຊ້ສໍາຮັບເປີດແພິມເພີ່ມເຕີມໃຫ້ກັບຜູ້ເຂົ້າໃນໂປຣແກຣມ คือ คำสั่ง with ທີ່ຈະ
ຄວາມສາມາດໃຊ້ໄດ້ເປີດແພິມໄດ້ຕະຫຼາດໄດ້ (ມີຄວາມສາມາດໃຊ້ໄດ້ຕະຫຼາດໄດ້) ແລະ
ເຄລືຍຮູ້ຂໍ້ມູນທີ່ອຸ່ນໃໝ່ໃນໜ່ວຍຄວາມຈຳແນບອັຕໂນມຕີ ທີ່ຈະມີຮູ່ແບນຄວາມສັ່ງຕີ່ຈະ

`with open(filename) as file`

โดย filename คือ ชื่อแฟ้มที่ต้องการเปิดใช้งาน, file คือ อ็อปเจกที่อ้างอิงไปยัง filename เพื่อนำไปใช้งาน ตัวอย่างการเปิดแฟ้มด้วยคำสั่ง with แสดงในโปรแกรมที่ 10.8

ตัวอย่างโปรแกรมที่ 10.8 แสดงการใช้คำสั่ง with

Program Example 10.8: with

```

1 # with with open file
2 # use with to read file
3 ↪3 with open("newfile.txt") as file
4 ↪4     for line in file:
5 ↪5         print (line)
6
7 #use with to write file
8 ↪8 with open("hello.txt", "w") as file:
9 ↪9     file.write("Hello World")

```



OUTPUT

```

hello world in the new file
and another line
Sequence string line 1
Sequence string line 2

```



OUTPUT

OUTPUT: hello.txt

```
Hello World
```

โปรแกรมที่ 10.8 เป็นการใช้คำสั่ง with ในการอ่านและเขียนแฟ้มข้อมูล บรรทัดที่ 3 โปรแกรมจะใช้คำสั่ง with ทำการสร้างอ็อปเจกที่อ้างอิงไปยังแฟ้มข้อมูลในหน่วยความจำavarชื่อ newfile.txt โดยใช้ชื่อสำหรับอ้างอิงชื่อ file บรรทัดที่ 4 โปรแกรมใช้คำสั่ง for เพื่ออ่านข้อมูลที่อ้างด้วยอ็อปเจก file ไปใช้งานครั้งละ 1 บรรทัดเก็บไว้ในตัวแปร line จากนั้นบรรทัดที่ 5 โปรแกรมทำการพิมพ์ข้อมูลที่อยู่ในตัวแปร line ออกจอภาพ โปรแกรมจะทำงานไปเรื่อยๆ จนกว่าข้อมูลจะหมดแฟ้ม

ลำดับถัดไปบรรทัดที่ 8 โปรแกรมจะทำการเปิดแฟ้มชื่อ hello.txt ในโหมดการเขียน บรรทัดที่ 9 โปรแกรมทำการเขียนข้อความว่า “Hello World” ลงไปในแฟ้มดังกล่าว เมื่อเปิดแฟ้ม hello.txt ที่อยู่ในไดเรคทรอรี่ปัจจุบันจะปรากฏผลลัพธ์ในตัวอย่างເຫຼົາຕຸພູດຕ້ານນັ້ນ ສັງເກດວ່າการใช้คำสั่ง with ໄນຈະເປັນຕົ້ນທີ່ກ່າວກັບການໃຊ້คำสั่ง open ໂດຍໄມ້ມີຄຳສັ່ງ with ກໍາກັນ

การจัดการแฟ้มและไดเรคทรอรี่

ภายในโมดูล os ของ Python มีฟังชันและเมธอดสำหรับบริหารจัดการเกี่ยวกับการประมวลผลแฟ้มข้อมูล (file processing) ไว้อย่างมากมาย เมื่อต้องการนำมาใช้งาน อันดับแรกจะต้องทำการ import เข้ามาในโปรแกรมเสียก่อน ลำดับต่อไปจึงเรียกใช้งานฟังชันเหล่านั้นได้

การเปลี่ยนชื่อแฟ้มข้อมูล มีรูปแบบคำสั่งคือ

```
os.rename("current_filename", "new_filename")
```

โดย current_filename คือชื่อแฟ้มปัจจุบันที่ต้องการเปลี่ยนชื่อ และ new_filename คือชื่อแฟ้มใหม่

การลบแฟ้มข้อมูล มีรูปแบบคำสั่งคือ

```
os.remove("filename")
```

โดย filename คือชื่อแฟ้มปัจจุบันที่ต้องการลบทิ้ง

การสร้างไดเรกทอรี่ มีรูปแบบคำสั่งคือ

```
os.mkdir("dirname")
```

โดย dirname คือ ชื่อดirekthor ที่ต้องการสร้างขึ้นใหม่

การเปลี่ยนที่อยู่ของไดเรกทอรี่ มีรูปแบบคำสั่งคือ

```
os.chdir("newname")
```

โดย newname คือ ชื่อดirekthor ที่ต้องการย้ายเข้าไปทำงาน

การแสดงไดเรกทอรี่ปัจจุบัน มีรูปแบบคำสั่งคือ

```
os.getcwd()
```

เมื่อตอนนี้ไม่จำเป็นต้องใช้พารามิเตอร์

การลบไดเรกทอรี่ มีรูปแบบคำสั่งคือ

```
os.rmdir("dirname")
```

โดย dirname คือ ชื่อดirekthor ที่ต้องการลบทิ้ง

สำหรับตัวอย่างการจัดการแฟ้มและไดเรกทอรี่ แสดงในโปรแกรมที่ 10.9

ตัวอย่างโปรแกรมที่ 10.9 ตัวอย่างการจัดการแฟ้มและไดเรกทอรี่

Program Example 10.9: file & directory management
--

```
1 # file and directory management
2 import os
3 # Rename a file from test1.txt to test2.txt
4 os.rename("test1.txt", "test2.txt")
5 # Delete file test2.txt
6 os.remove("test2.txt")
7 # Create a directory "C:\\test"
8 os.mkdir("C:\\test1")
9 # Changing a directory to "C:\\test1"
10 os.chdir("C:\\test1")
11 # This would give location of the current directory
12 print("Current directory is:",os.getcwd())
```

```
13 | # This would remove "C:\\test1" directory.  
14 | os.rmdir("C:\\test1")
```

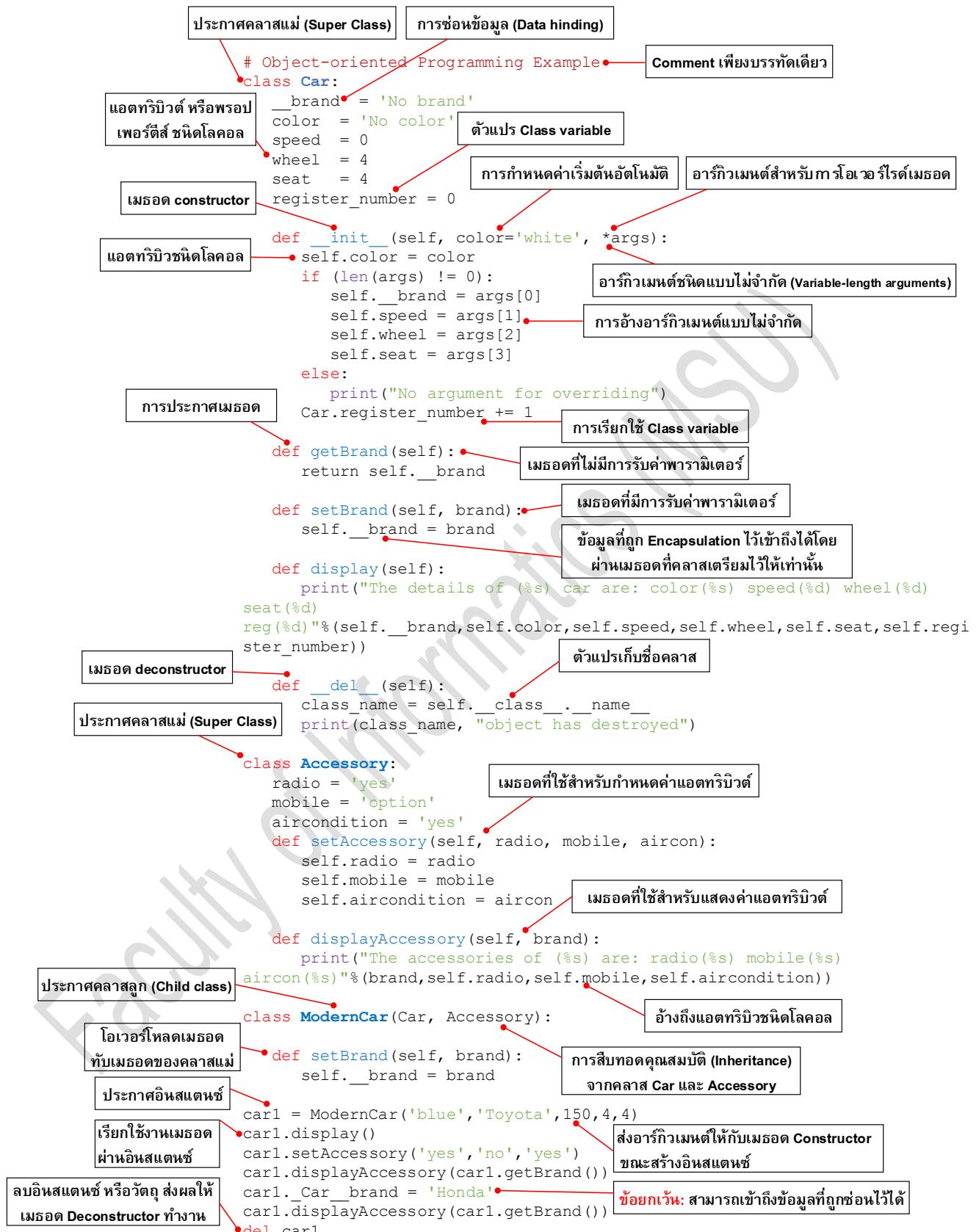


OUTPUT

Current directory is: C:\\test1

สำหรับการใช้งานพังชันและเมธอดเกี่ยวกับการจัดการแฟ้มและไดเรกทอรีเพิ่มเติม สามารถอ่านได้ในภาคที่ 5 เรื่อง Standard Library of Python

จบบทที่ 10

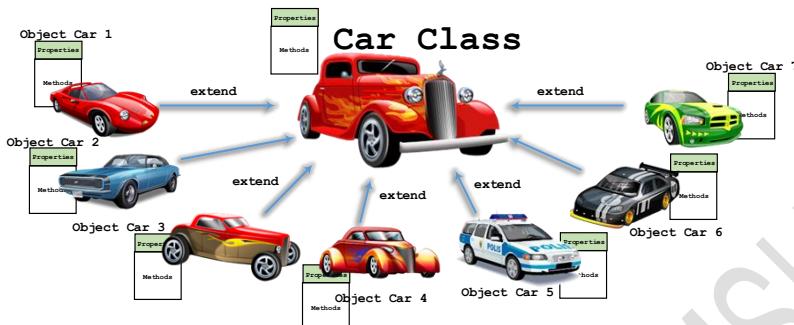


ภาพรวมการเขียนโปรแกรมเชิงวัตถุด้วยภาษาไพธอน

บทที่ 11

การเขียนโปรแกรมเชิงวัตถุ

(Object-Oriented Programming: OOP)



1. แนวคิดเกี่ยวกับหลักการเขียนโปรแกรม (Programming Paradigms)

แนวความคิดในการเขียนโปรแกรม คือ วิธีการพื้นฐานของการเขียนโปรแกรมคอมพิวเตอร์ ซึ่งสามารถจำแนกได้เป็น 4 ประเภทหลักๆ ได้แก่

- การเขียนโปรแกรมเชิงฟังก์ชัน (Functional programming) หรือเชิงโครงสร้าง (Structure programming) หรือแบบกระบวนการ (Procedure programming)
- การเขียนโปรแกรมเชิงคำสั่ง (Imperative programming)
- การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)
- การเขียนโปรแกรมเชิงตรรกะ (Logic programming)

นอกจากรูปแบบหลักทั้ง 4 แล้ว ยังมีอีกรูปแบบหนึ่งซึ่งขยายความสามารถของโมดูลโปรแกรม โดยใช้วิธีการตัดแทรกโปรแกรม เรียกว่า การโปรแกรมเชิงหน่วยย่อย (Aspect-Oriented programming)

ในบทเรียนก่อนๆ ที่ผ่านมาของหนังสือเล่มนี้ เป็นการกล่าวถึงการเขียนโปรแกรมเชิงฟังชัน ก่อนทั้งหมด แต่มีแทรกเกี่ยวกับการเขียนโปรแกรมเชิงวัตถุไว้บ้าง เช่น การเรียกใช้งาน Attribute เมธอด หรือการใช้งานคลาส เป็นต้น สำหรับในบทนี้จะเจาะลึกสำหรับการเขียนโปรแกรมเชิงวัตถุด้วยภาษาไทยตอน

2. แนวคิดเกี่ยวกับการโปรแกรมเชิงวัตถุ (Object-Oriented Programming Concept: OOP)

ก่อนจะเข้าสู่เนื้อหาการเขียนโปรแกรมเชิงวัตถุ จำเป็นต้องกล่าวถึงการเขียนโปรแกรมในรูปแบบเดิมๆ ก่อนว่าทำไงการเขียนโปรแกรมแบบเก่าจึงเริ่มถูกแทนที่ด้วยแนวคิดแบบเชิงวัตถุ การเขียน

โปรแกรมในรูปแบบเดิมมีจุดบกพร่องอย่างไร และแบบใหม่สามารถแก้ไขจุดบกพร่องเหล่านั้นได้จริง หรือไม่ ลองมาสำรวจความแตกต่างระหว่างการเขียนโปรแกรมทั้ง 2 แบบกันก่อน

แนวคิดการเขียนโปรแกรมเชิงฟังก์ชัน (Functional/Structure/Procedure programming)

จัดเป็นการเขียนโปรแกรมในรูปแบบเก่า มีลักษณะการเขียนโปรแกรม คือ การมองปัญหาหนึ่งๆ ออกเป็นส่วนย่อยๆ แล้วจึงค่อยแก้ไขไปทีละส่วนจนกว่าจะได้ผลลัพธ์ที่ต้องการ ซึ่งประยุกต์มาจากวิธีคิดของมนุษย์นั่นเอง เช่น เมื่อต้องการแก้ปัญหาโจทย์คณิตศาสตร์เรื่องสมการเชิงเส้น $Ax + By = C$ ถ้า มนุษย์คิดโดยปราศจากเครื่องคำนวณ จะแยกคำนวนประโยคคณิตศาสตร์นี้ทีละส่วน โดยเริ่มต้นจาก นำค่าคงที่ A คูณกับค่าในตัวแปร x เมื่อได้คำตอบแล้วหดไว้ในกระดาษก่อน ขั้นต่อไปคำนวน B คูณ y คำตอบที่ได้จะนำไปรวมกับ A คูณ x ผลรวมที่ได้จะถูกนำไปเปรียบเทียบกับค่าคงที่ C ในลำดับสุดท้าย สังเกตว่าปัญหาถูกแยกและทำงานเป็น 4 ขั้นตอน คือ

- 1) A คูณกับ x ผลลัพธ์ที่ได้ \Rightarrow พักไว้ในกระดาษหด
- 2) B คูณกับ y ผลลัพธ์ที่ได้ \Rightarrow พักไว้ในกระดาษหด
- 3) $Ax + By$ ผลลัพธ์ที่ได้ \Rightarrow พักไว้ในกระดาษหด
- 4) เปรียบเทียบค่า $Ax + By$ กับ C ผลลัพธ์ที่ได้ \Rightarrow เป็นจริงยุติการทำงาน ถ้าเป็นเท็จ กลับไปแก้ไข

ถ้าต้องการหาคำตอบจากสมการเชิงเส้นที่มีรูปแบบ $Ax + By = C$ จะต้องทำขั้นตอนในลักษณะ ดังกล่าวนี้เสมอๆ จึงเรียกว่า งานแบบฟังชัน (Function) หรือแบบกระบวนการ (Procedure) นั่นเอง สำหรับมุมมองในการพัฒนาโปรแกรม คำสั่งจะเรียงต่อกันไปเรื่อยๆ ทีละบรรทัด โปรแกรมจะเริ่มทำงาน จากคำสั่งแรกสุดเรื่อยไปจนถึงคำสั่งท้ายสุดเป็นอันจบโปรแกรม อาจมีการสร้างเป็นโปรแกรมย่อยๆ ใน โปรแกรมใหญ่บ้างเพื่อลดคำสั่งที่ซ้ำซ้อนลง

แนวคิดนี้มีข้อเสียอย่างไร? ขนาดของโปรแกรม (จำนวนบรรทัดของโปรแกรม หรือ lines of code) จะขึ้นอยู่กับความซับซ้อนของปัญหา ถ้าโปรแกรมมีขนาดและความซับซ้อนไม่มากจะไม่ก่อให้เกิด ปัญหาและไม่ยุ่งยากในการที่จะพัฒนาโปรแกรมด้วยวิธีนี้ ในทางตรงกันข้าม ถ้าปัญหามีขนาดใหญ่และ ซับซ้อนมากๆ จะทำให้การเขียนโปรแกรมซับซ้อนตามไปด้วย และยังพบปัญหาในเรื่องของการนำ โปรแกรมที่เขียนแล้วกลับมาใช้ใหม่ (Reusable) การแก้ไข (Modifying) การขยายเพิ่มเติม (Extensible) การบำรุงรักษา (Maintenance) ในระยะยาวอีกด้วย

แนวคิดเชิงวัตถุ (Object) คือ แนวคิดการพัฒนาโปรแกรมรูปแบบใหม่ที่นำมาใช้กันในปัจจุบัน เพื่อแก้ปัญหาดังกล่าวมาแล้วข้างต้นกับการการเขียนโปรแกรมแบบเชิงฟังชัน ถึงแม้รูปแบบการเขียนจะ ค่อนข้างยากและมีความซับซ้อน แต่จะส่งผลดีต่อการดูแลรักษาโปรแกรมในระยะยาว ซึ่งแนวคิดนี้จะ

แยกปัญหาหรือแยกระบบงานออกเป็นส่วนย่อย เช่น กัน เพื่อลดความซับซ้อนให้น้อยลง ส่วนย่อยหรือโปรแกรมย่อยเรียกว่า คลาส (Class) ภายในคลาสจะประกอบด้วยคุณสมบัติ (Properties/Attributes) และพฤติกรรมการตอบสนองกับสิ่งเล้าภายนอก (Behaviors/Methods)

ข้อดีของการเขียนโปรแกรมเชิงวัตถุ คือ

- เข้าใจง่าย เพราะการทำงานจะเลียนแบบสภาพแวดล้อมจริง โดยอาศัยการมองทุกอย่าง เป็นวัตถุ (Object) ที่มีหน้าที่และความหมายในตัว
- บำรุงรักษาและแก้ไขโปรแกรมได้ง่าย ลดผลกระทบต่อส่วนอื่นของโปรแกรม
- มีความปลอดภัยสูง เพราะจัดการกับความผิดพลาดของโปรแกรมได้ดี
- การซ่อนข้อมูล (Information hidden) และมีกลไกในการเข้าถึงข้อมูลอย่างปลอดภัย
- มีคุณสมบัติในการสืบทอด (Inheritance)
- นำกลับมาใช้ใหม่ได้ (Reusability) ลดขั้นตอนในการเขียนโปรแกรม
- โปรแกรมมีคุณภาพสูง ใช้ได้หลายแพลตฟอร์ม (platform)

ข้อเสียของการเขียนโปรแกรมเชิงวัตถุ คือ

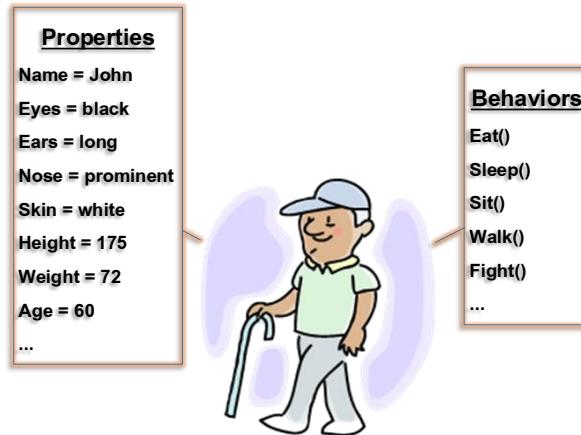
- เข้าใจจากสำหรับผู้เริ่มต้นเขียนโปรแกรมหรืออ่านด้วยตนเองเขียนโปรแกรมแบบเชิงฟังชันมาก่อน
- ทำงานได้ช้ากว่าภาษาโปรแกรมที่พัฒนาด้วยแนวความคิดแบบเชิงฟังชัน
- ภาษามีความกำหนด ถ้ามีลักษณะการสืบทอดที่ซับซ้อน (Multiple inheritance)

ความหมายการโปรแกรมเชิงวัตถุ (Object Oriented Programming)

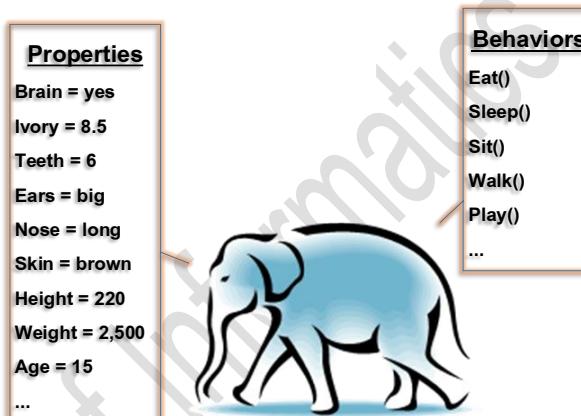
การโปรแกรมเชิงวัตถุหรือเรียกสั้นๆ ว่า OOP เป็นวิธีการเขียนโปรแกรมโดยมองสิ่งต่างๆ ในระบบเป็นวัตถุ (Object) ขึ้นหนึ่ง

วัตถุ คือ การมองภาพสิ่งต่างๆ เป็นวัตถุเบ้าหมาย โดยภายในวัตถุประกอบด้วยคุณสมบัติของวัตถุ (Properties หรือ Attributes) กับพฤติกรรมหรือการกระทำที่ตอบสนองต่อเหตุการณ์ต่างๆ (Behaviors) ตัวอย่างของวัตถุในสภาพความเป็นจริง เช่น

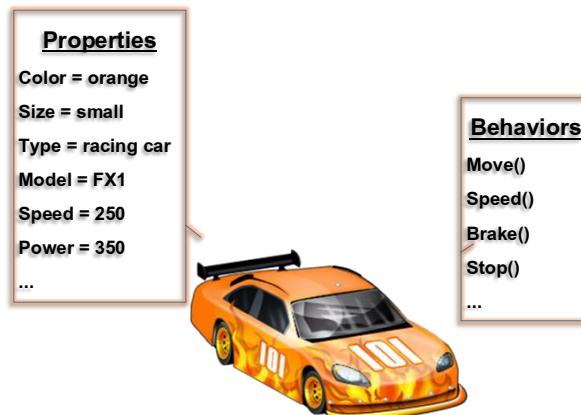
วัตถุ \Rightarrow มนุษย์ ประกอบด้วยคุณสมบัติ (Properties) คือ มีตา หู จมูก ลิ้น ผิวหนัง ผม สีผิว ความสูง สัดส่วน ความเป็นหญิง-ชาย สัญชาติ อายุ ชื่อ-นามสกุล เป็นต้น และมีพฤติกรรมการตอบสนองต่อเหตุการณ์ (Behavior) เช่น การกิน นอน นั่ง ยืน เดิน ต่อสู้ เป็นต้น



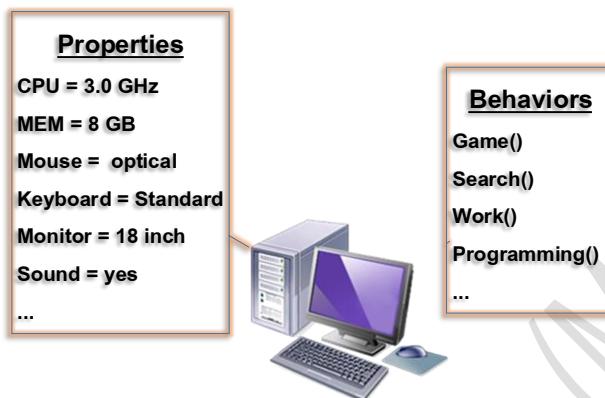
วัตถุ \Rightarrow ช้าง ประกอบด้วยคุณสมบัติ คือ มีสมอง พัน งาน ผิวนัง เล็บเท้า งวง หาง หู ตา ปาก ลำตัว เป็นต้น และมีพฤติกรรมการตอบสนองต่อเหตุการณ์ เช่น เดิน นอน กิน วิ่ง ขับถ่าย เล่น ผสม พันธุ์ ลากซุง เป็นต้น



วัตถุ \Rightarrow รถยนต์ ประกอบด้วยคุณสมบัติ คือ มีสีรถ ขนาด รุ่น ประเภท เชื้อเพลิงที่ใช้ ความเร็ว จำนวนที่นั่ง กำลังขับ เป็นต้น และมีพฤติกรรมการตอบสนองต่อเหตุการณ์ เช่น เคลื่อนที่ เร่ง ความเร็ว หยุดหรือเบรก เลี้ยว เป็นต้น



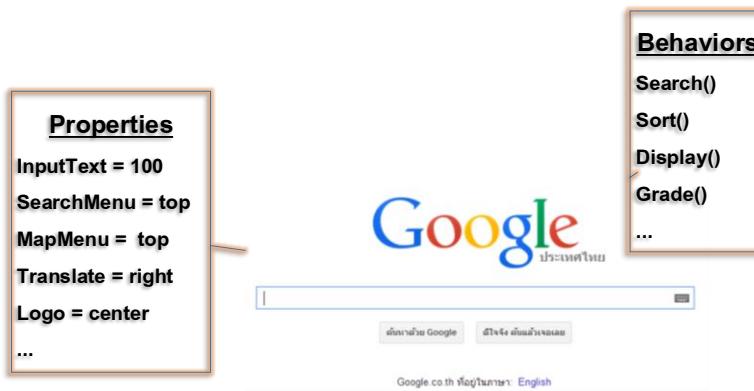
วัตถุ \Rightarrow คอมพิวเตอร์ ประกอบด้วยคุณสมบัติ คือ มีความรู้ หน่วยความจำ เมมส์ คีย์บอร์ด เสียง จอป้าพ เมนบอร์ด พัดลม ระบบปฏิบัติการ รุ่น สี ขนาด เป็นต้น และมีพฤติกรรมการตอบสนองต่อเหตุการณ์ เช่น เล่นเกมส์ พังเพลง เขียนโปรแกรม ชุมภาพยนต์ พิมพ์งาน ค้นหาข้อมูล ชมเว็บไซต์ chat เป็นต้น



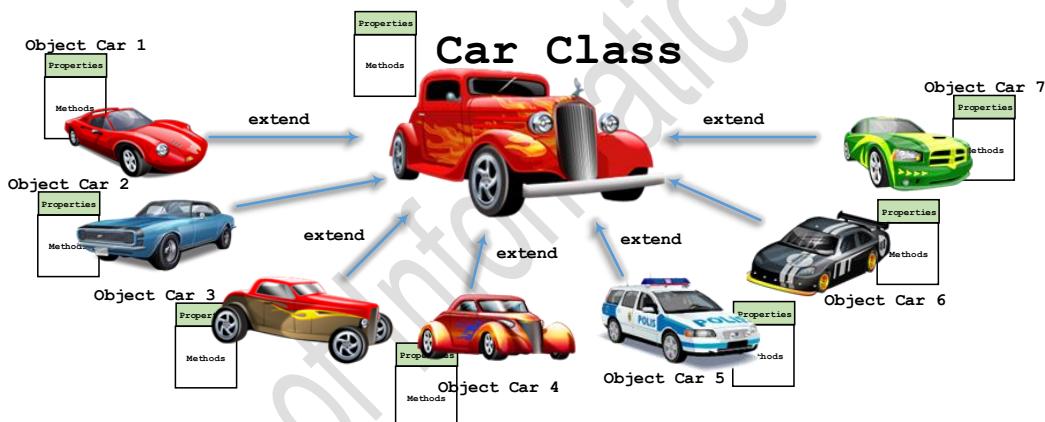
วัตถุ \Rightarrow ปุ่ม Enter บนคีย์บอร์ด ประกอบด้วยคุณสมบัติ คือ มีรูปร่าง ขนาด สี ภาษาหรือตัวอักษรบนปุ่ม เป็นต้น และมีพฤติกรรมการตอบสนองต่อเหตุการณ์ เช่น การกด การปล่อย การกดค้าง การสั่งงานโปรแกรมทำงาน การขึ้นบรรทัดใหม่ เป็นต้น



วัตถุ \Rightarrow Google search engine ประกอบด้วยคุณสมบัติ คือ ช่องสำหรับกรอกข้อมูล เมนูค้นหา แผนที่ gmail ปฏิทิน แปลภาษา โลโก้ ปุ่มค้นหา รูปร่างเว็บ ขนาดเว็บ สี ภาษา เป็นต้น และมีพฤติกรรมการตอบสนองต่อเหตุการณ์ เช่น การค้นหา การจัดเรียงรายที่ค้นหา การจัดอันดับเว็บ การเชื่อมโยงข้อมูล การแสดงผล เป็นต้น



การเขียนโปรแกรมแบบ OOP จะพยายามมองทุกสิ่งทุกอย่างในมุมของการโปรแกรมให้เป็นวัตถุ ก่อนที่จะสร้างขอบเขตขึ้นมาได้ต้องสร้างคลาสขึ้นมาก่อนเสมอ คลาสก็เปรียบเสมือนแม่แบบ แม่พิมพ์หรือพิมพ์เขียว ส่วนวัตถุ คือ สิ่งที่เกิดจากแม่พิมพ์ วัตถุที่เกิดจากคลาสเดียวกันจึงมีคุณสมบัติพื้นฐานเหมือนกัน ดังนั้นในการสร้างวัตถุต่างๆ ขึ้นมา ต้องอยู่ในคลาสไดคลาสหนึ่งเสมอ ซึ่งแสดงในรูปที่ 11.1



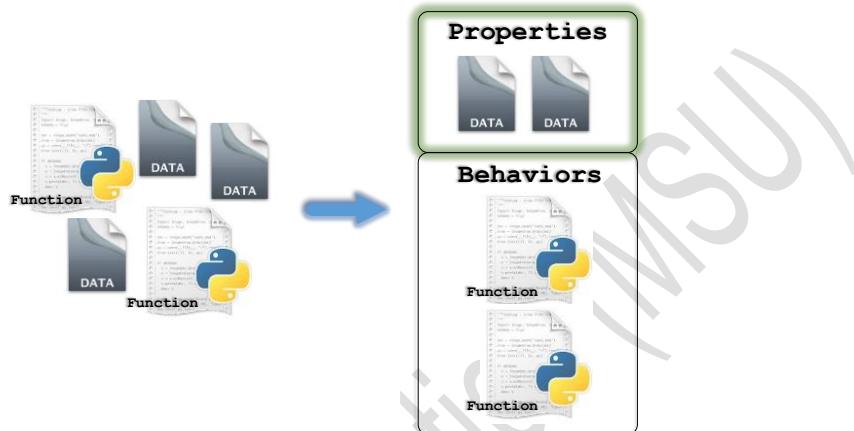
รูปที่ 11.1 แสดงการสร้างวัตถุจากคลาสรถยนต์

จากรูปที่ 11.1 แสดงให้เห็นถึงการสร้างวัตถุจากคลาสรถยนต์ (Car Class) ที่เปรียบเสมือนพิมพ์เขียวให้กับรถยนต์รุ่นใหม่ๆ ที่ถูกสร้างในเวลาต่อมา โดยคุณสมบัติหลักๆ จะถูกถ่ายทอดมาจาก Car Class เช่น จำนวนล้อ พวงมาลัย ระบบเกียร์ การทำงานของเครื่องยนต์ เชือเพลิงที่ใช้งาน ระบบไฟฟ้า เป็นต้น และพฤติกรรมที่ได้รับถ่ายทอดมา เช่น การเคลื่อนที่ การหยุด การเบรค การเลี้ยว การเปิดปิดระบบไฟฟ้า เป็นต้น แต่เมื่อสร้างเป็นวัตถุใหม่ เช่น Object Car 5 ซึ่งเป็นรถของตำรวจ อาจจะเพิ่มคุณสมบัติเฉพาะด้านขึ้นมาใหม่ได้ เช่น สี สัญญาณไฟ ความเร็ว อุปกรณ์วิทยุสื่อสาร เป็นต้น แต่ถ้าเป็นวัตถุชนิดรถแข่ง (Object Car 6) จะมีคุณสมบัติเพิ่มเติมพิเศษ เช่น ความเร็ว และระบบปรับอากาศความปลอดภัย เป็นต้น

ตัวดำเนินการ (Operation) หรือความหมายเดียวกับเมธอด (Method) นั้นเอง เป็นวิธีในการควบคุมหรือสั่งงานพฤติกรรมต่างๆ ของวัตถุ เพื่อตอบสนองต่อเหตุการณ์หรือคำสั่งตามที่เราต้องการ

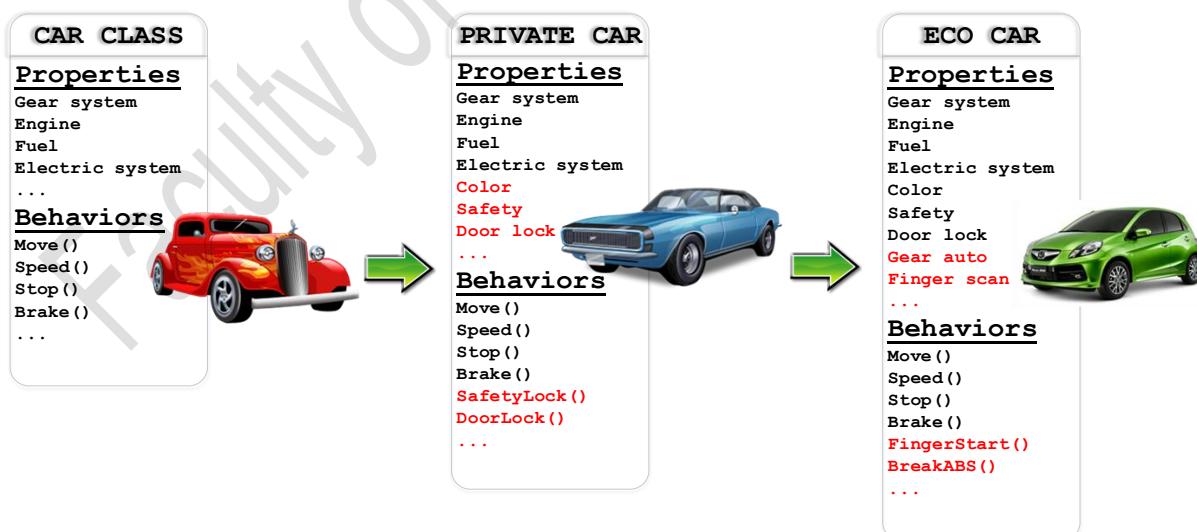
อินสแตนซ์ (Instance) คือ วัตถุที่ใช้งานจริงๆ ซึ่งถูกสร้างขึ้นในคลาสนั้นๆ เช่น วัตถุที่ถูกสร้างจากคลาสรถยนต์ (Car Class) ก็จะเป็นอินสแตนซ์ของคลาสรถยนต์นั้นเอง

การห่อหุ้มข้อมูล/การซ่อนข้อมูล (Encapsulation/Data hiding) คือ การห่อหุ้มข้อมูล โดยการรวมข้อมูลและฟังก์ชันเข้าไว้ในวัตถุตัวเดียวและมีกิลไก่ที่จะอนุญาตหรือไม่อนุญาตให้วัตถุอื่นๆ สามารถเข้าถึงได้ แสดงในรูปที่ 11.2



รูปที่ 11.2 แสดงจัดระเบียบข้อมูลและฟังก์ชันให้อยู่ในวัตถุเดียวกัน

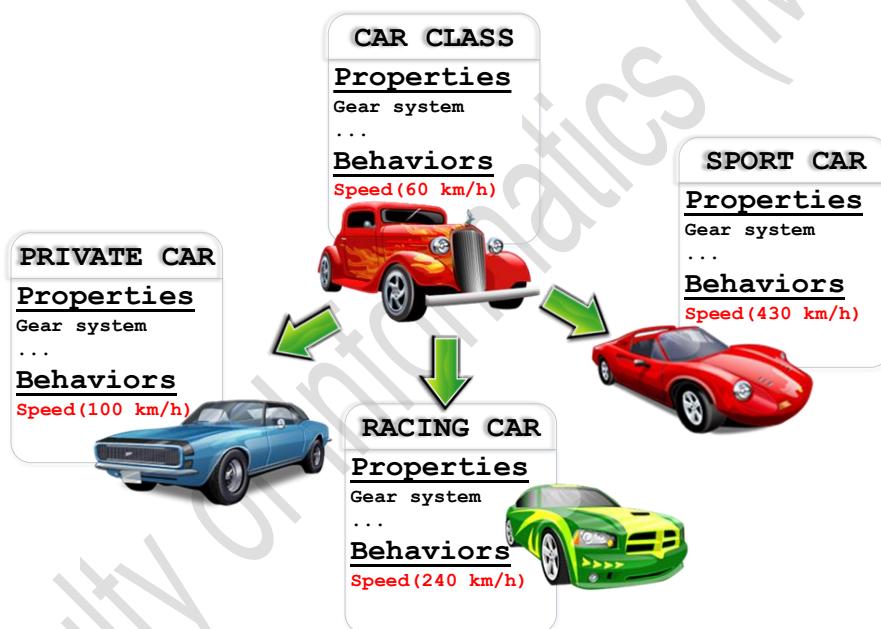
การสืบทอด (Inheritance) คือ คลาสแม่หรือคลาสหลัก (Super Class) สามารถสืบทอดคุณสมบัติต่างๆ ไปยังคลาสลูก (Sub Class) ได้ ซึ่งคลาสลูกจะรับคุณสมบัติทุกอย่างมาจากคลาสแม่ และสามารถเพิ่มเติมคุณสมบัติใหม่เฉพาะตัวของตนเข้าไปได้ (เป็นการขยายลักษณะพิเศษหรือความสามารถของวัตถุชนิดใหม่) ดังรูปที่ 11.3



รูปที่ 11.3 แสดงคุณสมบัติการสืบทอดรถยนต์

จากรูปที่ 11.3 เป็นตัวอย่างของการสืบทอดคุณสมบัติจากคลาสรถยนต์ (Car Class) ซึ่งมีคุณสมบัติพื้นฐาน เช่น ระบบเกียร์ นำมันเชือเพลิง เครื่องยนต์ ระบบไฟฟ้า และมีความสามารถในการเคลื่อนที่ หยุด เร่ง เบรค เป็นต้น เมื่อนำมาพิมพ์เขียวดังกล่าวมายกเป็นรถยนต์ชนิดส่วนตัว (Private Car) ก็จะได้รับคุณสมบัติพื้นฐานมาครบทั้งหมด แต่รถยนต์ส่วนบุคคลสามารถเพิ่มเติมคุณสมบัติพิเศษอื่นๆ เช่น เช่น สี ระบบปรับอากาศ ความปลอดภัย ระบบการล็อกประตู พื้นที่ใช้งาน เป็นต้น ต่างกันน้ำรถยต์ส่วนบุคคลมาเป็นต้นแบบ (Extend) โดยดัดแปลงประสิทธิภาพ เพื่อให้เหมาะสมกับสภาพในปัจจุบัน คือ รถยต์แบบ ECO Car โดยเพิ่มคุณสมบัติอื่นๆ เช่น เป็นเพื่อให้เหมาะสมกับสภาพการใช้งาน เช่น ระบบเกียร์แบบอัตโนมัติ ระบบสแกนลายนิ้วมือ ระบบเบรกอัตโนมัติ เป็นต้น

การพ้องรูป (Polymorphism) คือ เป็นคุณสมบัติของวัตถุใหม่ที่เกิดจากวัตถุแม่ชนิดเดียวกัน มีความสามารถเหมือนแม่แต่ผลลัพธ์การดำเนินงานไม่เหมือน คือ มีลักษณะเฉพาะตัว ดังรูปที่ 11.4



รูปที่ 11.4 แสดงคุณสมบัติการพ้องรูป

จากรูปที่ 11.4 แสดงความสามารถในการพ้องรูปของการพัฒนาโปรแกรมด้วย OOP จากตัวอย่าง คลาสรถยนต์ (Car Class) เป็นคลาสแม่ที่สามารถเพิ่มความเร็วในการเคลื่อนที่ได้สูงสุด คือ 60 กิโลเมตรต่อชั่วโมง เมื่อนำมาสร้างเป็นรถยนต์ส่วนบุคคล (Private) จะยังคงความสามารถในการวิ่งได้เหมือนคลาสแม่ แต่ทำความเร็วเพิ่มขึ้นอีก 40 กิโลเมตรต่อชั่วโมง (100 กิโลเมตรต่อชั่วโมง) แต่ถ้านำแบบไปสร้างเป็นรถแข่ง (Racing) จะสามารถเพิ่มความเร็วได้ถึง 240 กิโลเมตรต่อชั่วโมง แต่ถ้านำต้นแบบมาสร้างเป็นรถยนต์สปอร์ตจะมีความเร็วสูงสุดถึง 430 กิโลเมตรต่อชั่วโมง จากตัวอย่างนี้แสดงให้เห็นการพ้องรูปในเรื่องของความสามารถเร็วรถยนต์นั้นเอง

Class variable คือ ตัวแปรที่สามารถใช้งานร่วมกัน (Share) ระหว่างอินสแตนซ์และต่าง อินสแตนซ์ได้ โดยตัวแปร Class variable จะถูกกำหนดขึ้นภายในคลาสแต่อยู่นอกเมธอดของคลาส

เมธอด (Method) คือฟังก์ชันชนิดหนึ่งที่ถูกสร้างขึ้นภายในคลาส เพื่อใช้ดำเนินการกระทำที่ตอบสนองต่อเหตุการณ์ต่างๆ ของวัตถุ

แอตทริบิวต์/พรอพเพอร์ตี้/ดาต้า (Attributes/Properties/Data) คือ ตัวแปร ค่าคงที่ หรือ Class variable สำหรับเก็บข้อมูลที่มีความสัมพันธ์กับคลาสและวัตถุอยู่ภายในคลาส

คอนสตรัคเตอร์ (Constructor) คือ เมธอดหรือฟังก์ชันที่สร้างตัวเองโดยอัตโนมัติเมื่อมีการสร้างวัตถุหรืออินสแตนซ์จากคลาส มีเป้าหมายเพื่อเป็นการกำหนดสภาพแวดล้อมก่อนเริ่มต้นการทำงาน สำหรับไฟรอนมีชื่อเมธอด คือ `__init__()`

เดคอนสตรัคเตอร์ (Destructor) คือ เมธอดหรือฟังก์ชันที่ทำงานหลังจากมีการเรียกใช้คำสั่งลบวัตถุหรืออินสแตนซ์ (`del object`) มีเป้าหมายเพื่อเป็นการเคลียร์ค่าตัวแปรต่างๆ ก่อนคืนหน่วยความจำให้กับระบบ ชื่อเมธอด คือ `__del__()`

Overriding method/function คือ เมธอดหรือฟังก์ชันที่มีชื่อเหมือนกันกับเมธอดในคลาスマ่ แต่มีการดำเนินงานที่แตกต่างกัน

Overloading method/function คือ เมธอดหรือฟังก์ชันที่ชื่อเหมือนกัน แต่สามารถแยกการทำงานของแต่ละฟังก์ชันด้วยอาภิวัฒน์

Overloading operator ตัวดำเนินการที่ทำงานได้กับข้อมูลหลายชนิด

3. เริ่มต้นเขียนโปรแกรมเชิงวัตถุด้วยไฟรอน

การสร้างคลาส (Creating class)

การเขียนโปรแกรมเชิงวัตถุจะมองทุกสิ่งเป็นวัตถุ ดังนั้นก่อนที่จะสร้างวัตถุใดๆ ขึ้นมา จำเป็นต้องสร้างพิมพ์เขียวหรือคลาสขึ้นมาเสียก่อน (การสร้างคลาสจึงเป็นกระบวนการแรกของแนวคิดของการเขียนโปรแกรมเชิงวัตถุ) รูปแบบคำสั่งในการสร้างคลาสของภาษาไฟรอนเขียนได้ดังนี้ คือ

```
class ClassName:  
    'Optional class documentation string'  
    class_suite
```

การประกาศคลาสจะใช้คีย์เวิร์ดคือ `class` ตามด้วยชื่อของคลาส และปิดท้ายด้วย : (Colon)

'Optional class documentation string' คือ คำอธิบายเกี่ยวกับคุณลักษณะของคลาสที่สร้างขึ้น

สามารถเข้าถึงคำอธิบายดังกล่าวได้โดยใช้คำสั่ง `ClassName.__doc__` (คำอธิบายของคลาสอาจไม่มีก็ได้ : optional), `class_suite` เป็นส่วนของคำสั่งและคุณสมบัติต่างๆ ที่อยู่ภายใต้คลาส ประกอบไปด้วย สมบัติกของคลาส แอตทริบิวต์และเมธอดต่างๆ เป็นต้น สำหรับตัวอย่างการสร้างคลาสแสดงในโปรแกรมที่ 11.1

Program Example 11.1: creating class

```

1   # Creating car class(private attributes & methods)
2   class Car:
3       # attributes
4       color = "No brand yet"
5       brand = "No brand yet"
6       number_of_seats = 4
7       number_of_wheels = 4
8       maxSpeed = 0
9       registration_number = 0
10
11      def __init__(self, color, brand, number_of_seats,
12                  number_of_wheels, maxSpeed):
13          self.color = color
14          self.brand = brand
15          self.number_of_seats = number_of_seats
16          self.number_of_wheels = number_of_wheels
17          self.maxSpeed = maxSpeed
18          self.registration_number += 1
19
20      # methods
21      def setColor(self,x):
22          self.Color = x
23
24      def setBrand(self,x):
25          self.brand = x
26
27      def setNumberOfSeats(self,x):
28          self.number_of_seats = x
29
30      def setNumberOfWeels(self,x):
31          self.number_of_wheels = x
32
33      def setMaxSpeed(self,x):
34          self.maxSpeed = x
35
36      def printData(self):
37          print("The color of this car is :", self.color)
38          print("The car was manufactured by :", self.brand)
39          print("The number of seats is :",
40                self.number_of_seats, "seats.")
41          print("The number of wheels is :",
42                self.number_of_wheels, "wheels.")
43          print("The maximum speed is :", self.maxSpeed,
44                "km/h.")
45          print("The registration number is :",
46                self.registration_number)

```

```

42
43 # Creating instance and use it
⇒44 car1 = Car('red','Toyota',4,4,150)
45 car1.printData()
46 car1.color = 'Blue'
47 car1.printData()
48 car2 = Car('Yello','Honda',4,4,170)
49 car2.printData()

```



OUTPUT

```

The color of this car is : red
The car was manufactured by : Toyota
The number of seats is : 4 seats.
The number of wheels is : 4 wheels.
The maximum speed is : 150 km/h.
The registration number is : 1
The color of this car is : Blue
The car was manufactured by : Toyota
The number of seats is : 4 seats.
The number of wheels is : 4 wheels.
The maximum speed is : 150 km/h.
The registration number is : 1
The color of this car is : Yello
The car was manufactured by : Honda
The number of seats is : 4 seats.
The number of wheels is : 4 wheels.
The maximum speed is : 170 km/h.
The registration number is : 1

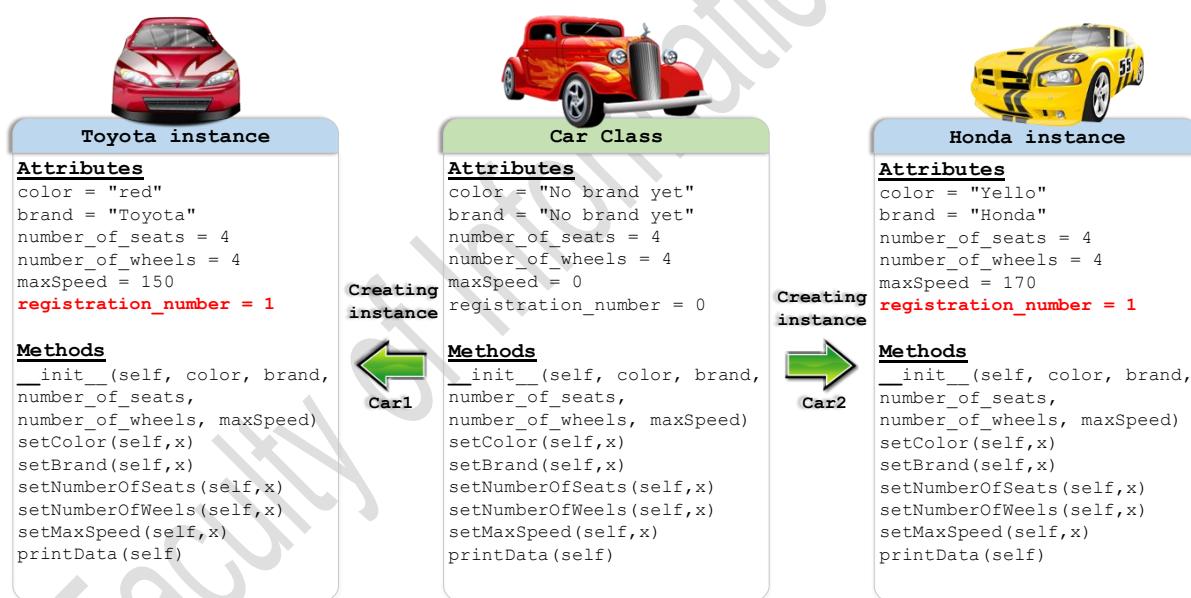
```

จากโปรแกรมที่ 11.1 และการประยุกต์ใช้ Car (บรรทัดที่ 2) โดยมีแอตทริบิวต์ 6 (บรรทัดที่ 3) ตัวประกอบด้วย color (สีของรถ) ไม่กำหนดค่าเริ่มต้น, brand (รุ่นหรือยี่ห้อ) ไม่กำหนดค่าเริ่มต้น, number_of_seats (จำนวนที่นั่ง) ค่าเริ่มต้นเท่ากับ 4, number_of_wheels (จำนวนล้อ) เท่ากับ 4, maxSpeed (ความเร็วสูงสุด) เท่ากับ 0 และ registration_number (ซึ่งเป็นการลงทะเบียนรถยนต์ที่สร้างไว้แล้วทั้งหมด จะถูกเพิ่มค่าครั้งละ 1 ทุกครั้งเมื่อมีการสร้างอินสแตนซ์ขึ้นใหม่) เท่ากับ 0 ตามลำดับ ต่อไปโปรแกรมจะทำการกำหนดค่าเริ่มต้นสำหรับตัวแปรต่างๆ (บรรทัดที่ 11) ด้วยเมธอดคอนสตรัคเตอร์ `__init__()` โดยนำอาร์กิวเมนต์ที่รับเข้ามากำหนดให้กับตัวแปรที่ประกาศไว้ การอ้างถึงตัวแปรในคลาสจะต้องขึ้นต้นด้วยคำว่า self หน้า เช่น `self.color` ซึ่งหมายถึงตัวแปรนี้จะถูกใช้งานภายในอินสแตนซ์เดียวกันเท่านั้น ไม่สามารถใช้งานร่วมกับอินสแตนซ์อื่นๆ ได้ เมธอดนี้จะทำงานเพียงครั้งเดียวเท่านั้นขณะสร้างอินสแตนซ์ของคลาส Car

ลำดับถัดไปโปรแกรมจะสร้างเมธอดชื่อ `setColor` (บรรทัดที่ 20) ซึ่งกำหนดค่าสีให้กับตัวแปร color โดยมีพารามิเตอร์ที่ใช้งาน 2 ตัวคือ self และ x ซึ่ง self คือการอ้างถึงตัวเอง (ชื่อข้อมูลภายในอินสแตนซ์ของตัวเอง) และ x คือ สีของรถยนต์ที่ต้องการกำหนด (เมื่อเรียกใช้งานเมธอดนี้ไม่ต้องส่งค่าให้กับพารามิเตอร์ self) ในคลาส Car มีหลายเมธอดที่กำหนดค่าให้กับตัวแปรต่างๆ ดังนี้คือ กำหนดยี่ห้อรถยนต์ `setBrand` (บรรทัดที่ 23), จำนวนที่นั่ง `setSeats` (บรรทัดที่ 26), จำนวน

ล้อ setWheels (บรรทัดที่ 29), ความเร็วสูงสุดที่สามารถทำความเร็วได้ setMaxSpeed (บรรทัดที่ 32) และแสดงข้อมูลรถยนต์ทั้งหมด printData (บรรทัดที่ 35)

เมื่อต้องการใช้งานคลาสดังกล่าวจะต้องทำการสร้างอินสแตนซ์ (บรรทัดที่ 44) ขึ้นก่อน (ซึ่งจะอธิบายอย่างละเอียดในหัวข้อถัดไป) พร้อมกับส่งค่าอาร์กิวเมนต์ให้กับคอนสตรัคเตอร์อย่างเหมาะสม เช่น car1 = Car('red', 'Toyota', 4, 4, 150) เมื่อสร้างอินสแตนซ์แล้ว จะใช้อินสแตนซ์ที่สร้างขึ้นเรียกใช้งานแอ็ตทริบิวต์และเมธอดที่ประกาศไว้ในคลาสได้ เช่น car.color = 'blue' หรือ car.setColor('Blue') เป็นต้น จากตัวอย่างโปรแกรมนี้ให้สังเกตที่ตัวแปร registration_number มีค่าเท่ากับ 1 ตลอดเวลาไม่เปลี่ยนแปลง ซึ่งในความเป็นจริงเมื่อมีการสร้างอินสแตนซ์ใหม่ๆ ก็จะมีค่าเดียวกัน ครั้ง ค่าดังกล่าวควรจะเพิ่มขึ้นไปตามจำนวนของรถยนต์ที่ถูกสร้างขึ้น (มีหมายเลขอหะเบียนรถที่ไม่ซ้ำกัน) ทั้งนี้เป็นเพราะว่าตัวแปรดังกล่าวมีขอบเขตการใช้งานเฉพาะในอินสแตนซ์ของตนเองเท่านั้น จะนั้นการเพิ่มค่า registration_number ในเมธอดคอนสตรัคเตอร์ __init__ จึงไม่มีผลกับการสร้างอินสแตนซ์ใหม่ แสดงในรูปที่ 11.5



รูปที่ 11.5 แสดงการสร้างอินสแตนซ์รถยนต์ Toyota และ Honda



Note: แอ็ตทริบิวต์ (Attribute) และคุณสมบัติ (Propertie) มีความหมายเดียวกัน ส่วนฟังชัน (Function) เมธอด (Method) หรือ พฤติกรรม (Behavior) มีความหมายไปในทำนองเดียวกัน การอธิบายในเชิงทฤษฎีนิยมใช้ คุณสมบัติ และ พฤติกรรมมากกว่า ส่วนในทางปฏิบัตินิยมใช้ แอ็ตทริบิวต์ และ เมธอดแทน

โปรแกรมตัวอย่างที่ 11.1 เป็นการสร้างคลาสที่ใช้แอ็ตทริบิวต์ชนิดโลคอล (Local) เท่านั้น คือ สามารถใช้งานเฉพาะภายในอินสแตนซ์เดียวกันเท่านั้น ไม่สามารถใช้ข้ามอินสแตนซ์ได้ ดังนั้นใน

โปรแกรมที่ 11.2 และรูปที่ 11.6 จะแสดงการประกาศและทริบิวต์ชนิด Variable class ซึ่งเป็นตัวแปรที่สามารถใช้งานร่วมกันระหว่างอินสแตนซ์ที่สร้างจากคลาสเดียวกัน

Program Example 11.2: *class variable*

```
1 # Creating Car class (Class variable)
2 class Car:
3     # attributes
4     color = "No brand yet"
5     brand = "No brand yet"
6     number_of_seats = 4
7     number_of_wheels = 4
8     maxSpeed = 0
9     registration_number = 0
10
11     def __init__(self, color, brand, number_of_seats,
12         number_of_wheels, maxSpeed):
13         self.color = color
14         self.brand = brand
15         self.number_of_seats = number_of_seats
16         self.number_of_wheels = number_of_wheels
17         self.maxSpeed = maxSpeed
18         Car.registration_number += 1
19
20     # methods
21
22     def printData(self):
23         print("The color of this car is :", self.color)
24         print("The car was manufactured by :", self.brand)
25         print("The number of seats is :",
26             self.number_of_seats, "seats.")
27         print("The number of wheels is :",
28             self.number_of_wheels, "wheels.")
29         print("The maximum speed is :", self.maxSpeed,
30             "km/h.")
31         print("The registration number is :",
32             self.registration_number)
33
34     # Creating instance and use it
35 car1 = Car('red','Toyota',4,4,150)
36 car1.printData()
37 car2 = Car('Yello','Honda',4,4,170)
38 car2.printData()
```



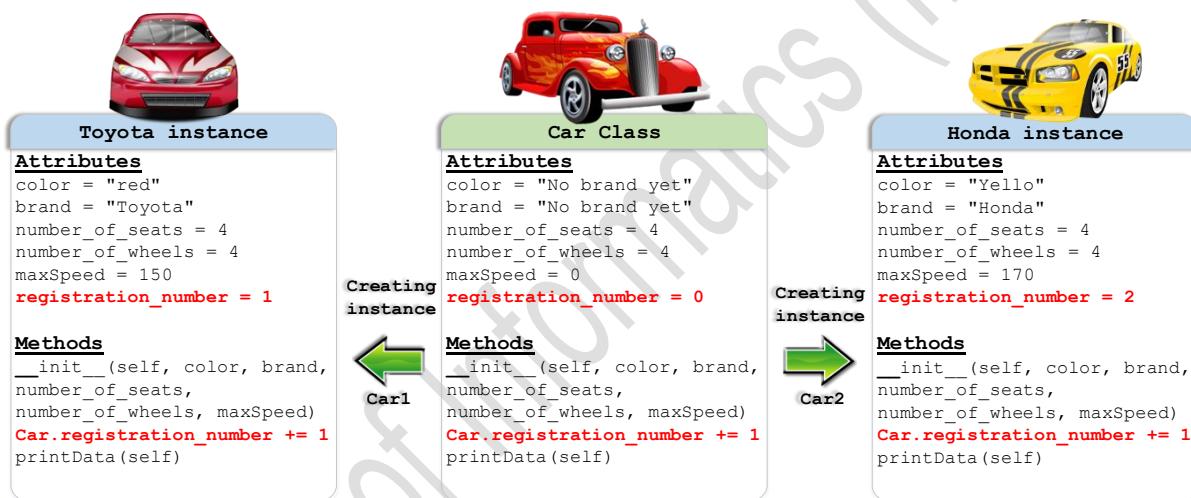
OUTPUT

The color of this car is : red
The car was manufactured by : Toyota
The number of seats is : 4 seats.
The number of wheels is : 4 wheels.
The maximum speed is : 150 km/h.
The registration number is : 1
The color of this car is : Yello
The car was manufactured by : Honda
The number of seats is : 4 seats.
The number of wheels is : 4 wheels.

The maximum speed is : 170 km/h.

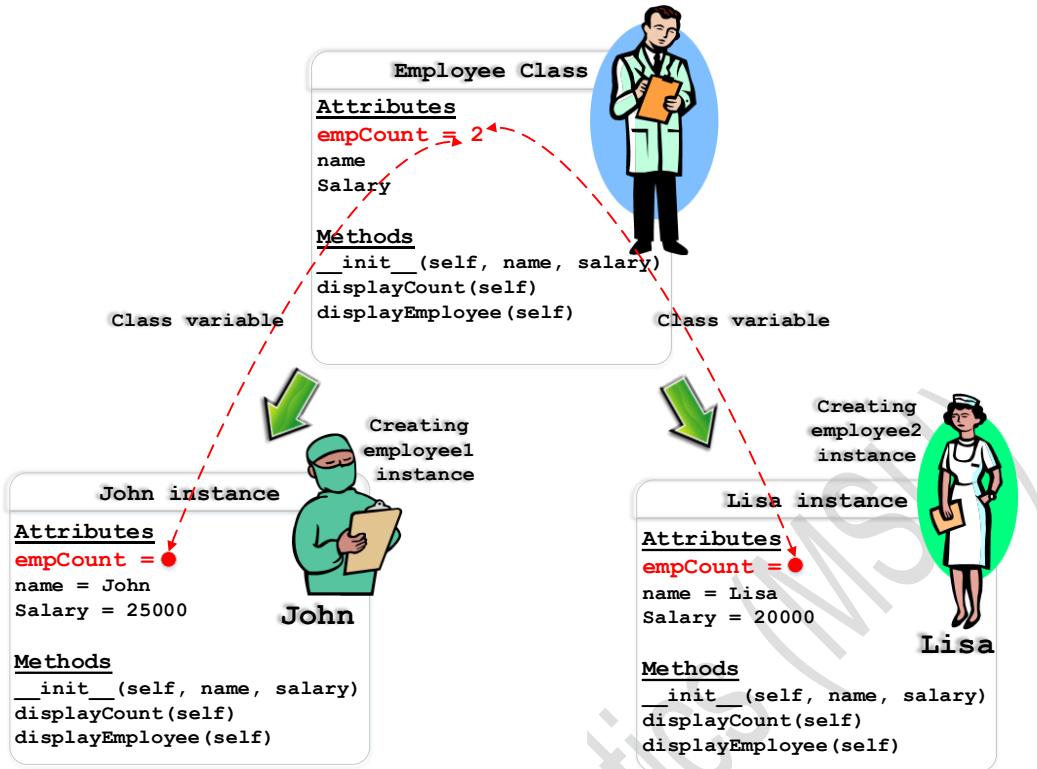
The registration number is : 2

จากตัวอย่างโปรแกรมที่ 11.2 มีความตั้งใจให้ตัวแปร `registration_number` เป็นชนิด Class variable (บรรทัดที่ 9) ซึ่งเมื่อพิจารณาคร่าวๆ แล้ว เมื่อนับตัวแปรรถมาทั้งไปไม่ได้มีความพิเศษ แตกต่างจากตัวแปรอื่นๆ เช่น `color` หรือ `brand` เป็นต้น แต่เมื่อสร้างอินสแตนซ์ เมื่อดคุณสตรัคเตอร์ จะเปลี่ยนคุณสมบัติของตัวแปร `registration_number` เป็นตัวแปรชนิด Class variable โดยใช้การอ้างชื่อคลาส `Car` แทน `self` (บรรทัดที่ 17) เช่น `Car.registration_number += 1` เมื่อสร้างอินสแตนซ์ตัวแรก จะทำให้ตัวแปร `registration_number` เพิ่มค่าจาก 0 เป็น 1 (บรรทัดที่ 30) และเมื่อสร้างอินสแตนซ์ที่สอง ค่าในตัวแปร `registration_number` จะเพิ่มเป็น 2 (บรรทัดที่ 32) ดังแสดงในรูปที่ 11.6 แสดงภาพของการสร้างอินสแตนซ์ `Car1` และ `Car2` ส่งผลให้ตัวแปร `registration_number` เพิ่มค่าเป็น 2 (เมื่อสร้างรถยกตัวมาแล้ว 2 คันหรือเลขทะเบียนรถยกตัวไม่ซ้ำกัน 2 หมายเลขนั้นเอง)



รูปที่ 11.6 แสดงการใช้งานตัวแปร Class variable (`registration_number`)

เพื่อให้เห็นภาพการสร้างคลาสและการใช้งานตัวแปรให้ชัดเจนขึ้น ผู้เขียนจะขอยกตัวอย่างการสร้างคลาสพนักงาน (Employee) อีกสักตัวอย่างดังรูปที่ 11.7 และโปรแกรมตัวอย่างที่ 11.3



รูปที่ 11.7 แสดงการสร้างอินสแตนซ์พนักงานสำหรับ John และ Lisa

Program Example 11.3: creating employee class

```

1      # Creating instance objects of Employee class
⇒2    class Employee:
⇒3        'To declear super class Employee for all employee'
⇒4        empCount = 0
5
⇒6        def __init__(self, name, salary):
7            self.name = name
8            self.salary = salary
9            Employee.empCount += 1
10
⇒11       def displayCount(self):
12           print ("Total Employee =%d" % Employee.empCount)
13
⇒14       def displayEmployee(self):
15           print ("Name : ", self.name, ", Salary: ",
self.salary)
16
17   #Creating instance objects
⇒18   employee1 = Employee("John", 25000)
19   employee2 = Employee("Lisa", 20000)
20   employee1.displayEmployee()
21   employee2.displayEmployee()
22   employee1.displayCount()

```



OUTPUT
Name : John , Salary: 25000
Name : Lisa , Salary: 20000
Total Employee = 2

จากตัวอย่างโปรแกรมที่ 11.3 แสดงการสร้างคลาสพนักงาน (Employee) เริ่มจากประกาศว่า เป็น class ชื่อ Employee (บรรทัดที่ 2) ในบรรทัดถัดมาเป็นการเขียนคำอธิบายเกี่ยวกับคลาสที่สร้างว่า มีหน้าที่อย่างไร (บรรทัดที่ 3) บรรทัดต่อไปเป็นการประกาศตัวแปรชนิด Class variable (บรรทัดที่ 4) ชื่อ empCount ซึ่งเป็นตัวแปรที่ใช้นับจำนวนของพนักงานทั้งหมด โดยตัวแปรดังกล่าวจะใช้งานร่วมกันระหว่างอินสแตนซ์ที่สร้างจากคลาสหนังงาน บรรทัดถัดไปเป็นเมธอดพิเศษ (บรรทัดที่ 6) ที่เรียกว่า คอนสตรัคเตอร์ (Constructor) ชื่อว่า __init__() ทำหน้าที่กำหนดค่าเริ่มต้นเพื่อเตรียมความพร้อม สำหรับคลาสพนักงาน เมธอดดังกล่าวจะทำงานเมื่อมีการสร้างอินสแตนซ์ใหม่จากคลาสพนักงาน เมธอดนี้มีพารามิเตอร์ 3 ตัวคือ self, name และ salary โดย self หมายถึงตัวคลาสพนักงานเอง (ชี้ตัวเอง) ผู้เขียนโปรแกรมไม่จำเป็นต้องส่งค่ากิวเมนต์ให้กับ self เช่นماayan เมธอดคอนสตรัคเตอร์ ตัวแปร name คือ ชื่อพนักงาน และ salary คือเงินเดือนของพนักงานแต่ละคน

การประกาศเมธอดในไฟล์จะประกาศเหมือนการประกาศฟังชันตามที่ได้เคยศึกษามาในบท ก่อนๆ อาร์กิวเมนต์ที่ส่งเข้ามายังคอนสตรัคเตอร์จะถูกกำหนดให้กับตัวแปร name (การใช้งานตัวแปรใน คอนสตรัคเตอร์ต้องอ้างด้วย self ก่อนเสมอ) คือ self.name = name เช่นเดียวกับ salary แต่มีข้อสังเกต ว่าการอ้างถึงตัวแปร empCount จะอ้างชื่อคลาสก่อน ไม่ใช่ self ทั้งนี้เพราะว่า คีย์เวิร์ด self หมายถึง ข้อมูลภายในขอบเขตของเมธอดเท่านั้น แต่ถ้าต้องการอ้างไปยังตัวแปรนอกเมธอดที่เป็นตัวแปรชนิด Class variable ต้องอ้างชื่อคลาสก่อน คือ Employee.empCount ตัวแปรนี้จะเก็บจำนวนของพนักงานเมื่อมี การสร้างอินสแตนซ์ใหม่ทุกๆ ครั้ง (คล้ายกับการรับพนักงานใหม่เข้ามาในบริษัท ถ้ามีการสร้าง อินสแตนซ์ 3 อินสแตนซ์ จะทำให้ค่า empCount = 3)

บรรทัดถัดไป (บรรทัดที่ 11) เป็นเมธอด DisplayCount โดยมีพารามิเตอร์ self ตัวเดียวเท่านั้น สำหรับหน้าที่ของเมธอนี้คือ พิมพ์จำนวนสมาชิกทั้งหมดของพนักงานและเมธอด DispalyEmployee (บรรทัดที่ 14) จะพิมพ์ชื่อพร้อมกับเงินเดือนทั้งหมดของพนักงาน จากนั้นโปรแกรมจะสร้างอินสแตนซ์ employee1 ชื่อ John และได้รับเงินเดือนเท่ากับ 25,000 ส่วน employee2 คือ Lisa และได้รับเงินเดือน 20,000 ตามลำดับ



Note: การตั้งชื่อแฟ้ม ควรตั้งให้ตรงกับชื่อคลาสที่สร้างขึ้น เช่น คลาส Employee ควรตั้งชื่อ แฟ้มเป็น Employee.py

การสร้างอินสแตนซ์ของวัตถุ (Creating instance objects)

การสร้างอินสแตนซ์สำหรับวัตถุ สามารถสร้างได้โดยการระบุชื่อคลาสต้นแบบและใส่ อาร์กิวเมนต์ให้ตรงกับเมธอดที่คอนสตรัคเตอร์ (`__init__()`) กำหนดไว้ โดยไม่ต้องส่งค่าให้กับอาร์กิวเมนต์ self ดังตัวอย่างโปรแกรมที่ 11.4

Program Example 11.4: creating instances of Object

```

1 #Creating instance objects
⇒2 employee1 = Employee("John", 25000)
⇒3 employee2 = Employee("Lisa", 20000)
⇒4 employee1.displayEmployee()
⇒5 employee2.displayEmployee()
⇒6 employee1.displayCount()
⇒7 employee1.empCount = 5
⇒8 employee1.salary = 30000
⇒9 employee2.salary = 28000
⇒10 employee1.displayEmployee()
⇒11 employee2.displayEmployee()
⇒12 employee2.displayCount()

```


OUTPUT

```

Name : John , Salary: 25000
Name : Lisa , Salary: 20000
Total Employee = 2
Name : John , Salary: 30000
Name : Lisa , Salary: 28000
Total Employee = 2

```

แสดงตัวอย่างการสร้างอินสแตนซ์ของวัตถุ โดยอ้างอิงจากตัวอย่างโปรแกรมที่ 11.3 สำหรับโปรแกรมที่ 11.4 เริ่มต้นจากบรรทัดที่ 2 โปรแกรมทำการสร้างอินสแตนซ์ชื่อ employee1 โดยกำหนดอาร์กิวเม้นต์ 2 ตัวคือ name = John และ salary = 25000 ต่อจากนั้นทำการสร้างอินสแตนซ์ employee2 (บรรทัดที่ 3) โดยกำหนดอาร์กิวเม้นต์ 2 ตัวเช่นเดียวกัน คือ name = Lisa และ salary = 20000 การสร้าง employee1 จะส่งผลให้คอนสตรัคเตอร์กำหนดชื่อ John และเงินเดือนเท่ากับ 25000 เก็บไว้ในตัวแปร name และ salary ตามลำดับ พร้อมกับทำการเพิ่มค่าของ empCount เป็น 1 เมื่อ employee2 ถูกสร้างคอนสตรัคเตอร์จะกำหนด name = Lisa และ salary = 20000 และเพิ่มค่า empCount เป็น 2 ตามลำดับ ในบรรทัดที่ 4, 5 โปรแกรมเรียกเมธอด displayEmployee() ของ อินสแตนซ์ employee1 และ employee2 เพื่อแสดงข้อมูล ผลลัพธ์ที่ได้คือ "Name : John , Salary: 25000" และ "Name : Lisa , Salary: 20000" ตามลำดับ

ต่อจากนั้นในบรรทัดที่ 6 โปรแกรมเรียกเมธอด displayCount() ซึ่งจะให้ผลลัพธ์เป็น 2 เนื่องจากสร้างพนักงานไปแล้วจำนวน 2 คนนั้นเอง ต่อจากนั้นโปรแกรมบรรทัดที่ 7 จะทดสอบ กำหนดค่าตัวแปร Clas variable คือ cmpCount ให้มีค่าเท่ากับ 5 พร้อมกับกำหนดเงินเดือนใหม่ให้กับ John เท่ากับ 30,000 และ Lisa เท่ากับ 28,000 ในบรรทัดที่ 8 และ 9 ต่อจากนั้นทำการพิมพ์ข้อมูลของ John และ Lisa พร้อมทั้งแสดงข้อมูลของ empCount (บรรทัดที่ 10, 11, 12) ผลลัพธ์ที่ได้แสดงให้เห็นว่า เงินเดือนใหม่ที่กำหนดให้จะสามารถปรับปรุงได้ แต่ค่าของ empCount จะไม่สามารถกำหนดเข้าไปได้ โดยตรง ต้องกำหนดผ่านเมธอดเท่านั้น ซึ่งจำกกล่าวในหัวข้อต่อไป

การเข้าถึงและเมธอด (Accessing attributes and methods)

การเข้าถึงและเมธอดของวัตถุสามารถทำได้โดยการอ้างชื่ออินสแตนซ์ของวัตถุตามด้วย สัญลักษณ์ . และชื่อของเมธอดหรือแอ็ตทริบิวต์ แต่ถ้าเป็นตัวแปรชนิด Class variable ต้องอ้างโดยใช้ชื่อคลาส ซึ่งมีรูปแบบดังนี้

การเข้าถึงและเมธอดของวัตถุแบบปกติ

Instance of Object.attribute
เช่น car1.color หรือ employee1.name

การเข้าถึงเมธอดของวัตถุแบบปกติ

Instance of Object.method()
เช่น car1.setColor() หรือ employee1.displayEmployee()

การเข้าถึงและเมธอดชนิด Class variable

Class.class_variable
เช่น Car.register_number หรือ Employee.empCount

ไฟชอนอนุญาตให้ผู้เขียนโปรแกรมสามารถเพิ่ม แก้ไข และลบแอ็ตทริบิวต์ของวัตถุที่ไม่ได้สร้าง เอาไว้ล่วงหน้าเพิ่มเติมได้ทันทีหลังจากสร้างอินสแตนซ์แล้ว เช่น

```
car1.fuel = "binzine"
```

```
employee1.age = 35
```

แอ็ตทริบิวต์ fuel (นำมันเชื่อเพลิง) และ age (อายุ) ไม่เคยถูกประกาศเป็นตัวแปรในคลาส Car และ Employee มาก่อน แต่เมื่อสร้างอินสแตนซ์ของวัตถุแล้วสามารถเพิ่มเติมแอ็ตทริบิวต์ต่างๆ เหล่านี้ได้ในภายหลัง แต่ผู้เขียนไม่แนะนำให้ทำถ้าไม่จำเป็น เพราะจะทำให้สับสนได้ว่าแอ็ตทริบิวต์ที่ประกาศเพิ่มเติมมีความจำเป็นกับคลาสที่สร้างขึ้นจริงหรือไม่ เพราะการประกาศเพิ่มเติมด้วยวิธีนี้แอ็ตทริบิวต์จะไม่ปรากฏอยู่ในคลาส สำหรับการลบแอ็ตทริบิวต์ที่สร้างขึ้นให้คำสั่ง del และตามด้วยแอ็ตทริบิวต์ที่ต้องการลบ เช่น

```
del car1.fuel
```

```
del employee1.age
```

การเข้าถึงและเมธอดที่ได้กล่าวมาแล้วเป็นวิธีการแบบปกติ แต่ไฟชอนมีวิธีการเข้าถึงและกำหนดค่าให้แอ็ตทริบิวต์เหล่านี้ได้ด้วยฟังชัน getattr(), hasattr(), setattr() และ delattr() ดังนี้

การเข้าถึงแอตทริบิวต์ด้วยคำสั่ง `getattr(obj, name[, default])` โดย `obj` หมายถึงวัตถุที่ใช้การอ้างอิง, `name` คือชื่อของแอตทริบิวต์ที่ต้องการใช้งาน ผลลัพธ์ที่ส่งกลับมาจากการฟังชันคือ ค่าที่อยู่ในแอตทริบิวต์ที่ระบุ ตัวอย่างเช่น

```
getattr(car1, 'fuel') หรือ getattr(employee1, 'age')
```

ผลลัพธ์จาก `fuel` คือ `benzine` และ `age` คือ `35`

การตรวจสอบแอตทริบิวต์ว่ามีอยู่หรือไม่ ด้วยคำสั่ง `hasattr(obj, name)` โดย `obj` หมายถึงวัตถุที่ใช้การอ้างอิง, `name` คือชื่อของแอตทริบิวต์ที่ต้องการใช้งาน ผลลัพธ์ที่ส่งกลับมาจากการฟังชันคือ เป็นจริงเมื่อแอตทริบิวต์ที่ระบุปรากฏอยู่ ถ้าไม่ปรากฏจะเป็นเท็จ ตัวอย่างเช่น

```
hasattr(car1, 'fuel') หรือ hasattr(employee1, 'sex')
```

ผลลัพธ์จาก `fuel` คือ `True` และ `sex` คือ `False` (เพราะไม่ได้เพิ่มแอตทริบิวต์ `sex` ไว้)

การทำหนด (Set) แอตทริบิวต์ ด้วยคำสั่ง `setattr(obj, name, value)` โดย `obj` หมายถึงวัตถุที่ใช้การอ้างอิง, `name` คือชื่อของแอตทริบิวต์ที่ต้องการกำหนดค่า และ `value` คือค่าที่ต้องการกำหนดให้แอตทริบิวต์ `name` ถ้าแอตทริบิวต์ดังกล่าวไม่มีอยู่ พังชัน `setattr` จะสร้างให้ใหม่ทันที ตัวอย่างเช่น

```
setattr(car1, 'fuel', 'diesel') หรือ setattr(employee1, 'age', 8)
```

การลบแอตทริบิวต์ ด้วยคำสั่ง `delattr(obj, name)` โดย `obj` หมายถึงวัตถุที่ใช้การอ้างอิง, `name` คือชื่อของแอตทริบิวต์ที่ต้องการลบ ตัวอย่างเช่น

```
delattr(car1, 'fuel') หรือ delattr(employee1, 'age')
```

แอตทริบิวต์ชนิด built-in ในคลาส (Built-In class attributes)

เมื่อผู้เขียนโปรแกรมสร้างคลาสโดยคลาสนั้นนำเข้ามาใช้งาน ไฟรอนจะสร้างแอตทริบิวต์ที่เรียกว่า `built-in attribute` มาให้พร้อมกับคลาสดังกล่าวทันที ซึ่งแอตทริบิวต์ชนิด `built-in` นี้จะมีหน้าที่อธิบายโครงสร้างของคลาสและรายละเอียดของคลาส ดังต่อไปนี้

`__dict__` เป็นแอตทริบิวต์ที่ทำหน้าที่แสดงรายละเอียดต่างๆ ที่อยู่ใน namespace ของคลาส

`__doc__` เป็นแอตทริบิวต์ที่ใช้อธิบายหน้าที่ของคลาสตามที่ผู้เขียนได้สร้างขึ้น (ในส่วน `class document string` ซึ่งอยู่หลังจากการประกาศชื่อคลาส)

`__name__` เป็นแอตทริบิวต์ที่ใช้เก็บชื่อคลาส

`__module__` เป็นแอตทริบิวต์ที่ใช้เก็บรายชื่อของโมดูลที่คลาสได้กำหนดไว้ แอตทริบิวต์ดังกล่าวจะเก็บฟังชัน `__main__` เมื่อยกในโหมด Interactive (Python shell)

`__bases__` (ไม่มีใน Python เวอร์ชัน 3.0) แต่มีในเวอร์ชันที่ต่ำกว่า ทำหน้าที่เก็บรายชื่อของคลาสพื้นฐานต่างๆ สำหรับตัวอย่างการใช้งานแอตทริบิวต์แบบ built-in แสดงในโปรแกรมที่ 11.5

Program Example 11.5: built-in class attributes

```

1  # Testing built-in attributes
2  class Employee:
3      'To declear super class Employee for all employee'
4      empCount = 0
5
6      def __init__(self, name, salary):
7          self.name = name
8          self.salary = salary
9          Employee.empCount += 1
10
11     def displayCount(self):
12         print ("Total Employee = %d" % Employee.empCount)
13
14     def displayEmployee(self):
15         print ("Name : ", self.name, ", Salary: ",
16               self.salary)
17
18     def setEmpCount(self, x):
19         Employee.empCount = x
20
#To access built-in attributes
⇒21 print("Employee.__name__: ",Employee.__name__)
⇒22 print("Employee.__doc__: ",Employee.__doc__)
⇒23 print("Employee.__module__: ",Employee.__module__)
⇒24 print("Employee.__dict__: ",Employee.__dict__)

```



OUTPUT

```

Employee.__name__: Employee
Employee.__doc__: To declear super class Employee for all
employee
Employee.__module__: __main__
Employee.__dict__: {'empCount': 0, '__dict__': <attribute
 '__dict__' of 'Employee' objects>, '__module__':
 '__main__', 'setEmpCount': <function Employee.setEmpCount
 at 0x000000003EF52F0>, '__doc__': 'To declear super class
Employee for all employee', 'displayCount': <function
Employee.displayCount at 0x000000003EF51E0>,
'displayEmployee': <function Employee.displayEmployee at
0x000000003EF5268>, '__init__': <function
Employee.__init__ at 0x000000000112D7B8>, '__weakref__':
<attribute '__weakref__' of 'Employee' objects>}

```

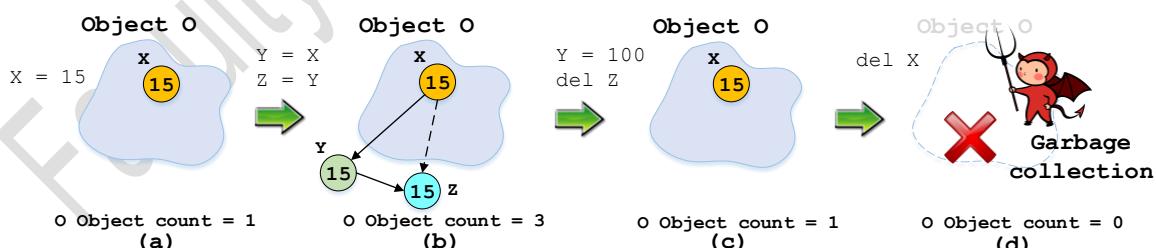
จากตัวอย่างโปรแกรมที่ 11.5 แสดงการเรียกใช้งานแอตทริบิวต์ชนิด built-in ในคลาส

Employee สำหรับการเรียกใช้งานจะต้องอ้างชื่อของคลาสตามด้วยแอตทริบิวต์ที่ต้องการ เช่น แอตทริบิวต์ `__name__` (บรรทัดที่ 21) เก็บชื่อของคลาส สำหรับในตัวอย่างนี้ชื่อ Employee, `__doc__` เก็บ

ข้อความใน class document string (บรรทัดที่ 22) ที่ผู้เขียนโปรแกรมเขียนไว้หลังการประกาศชื่อคลาส (บรรทัดที่ 3) มีค่าเท่ากับ “To declear super class Employee for all”, `__module__` เก็บชื่อของฟังชัน `main` (บรรทัดที่ 23) และ `__dict__` (บรรทัดที่ 24) เก็บรายละเอียดทั้งหมดที่เก็บอยู่ใน namespace ของคลาส `Employee`

การลบวัตถุที่สร้างขึ้น (Destroying objects)

ไฟรอนสนับสนุนกลไกการคืนหน่วยความจำแบบอัตโนมัติที่เรียกว่า Garbage collection ซึ่งกลไกดังกล่าวทำหน้าที่ลบอ้อมเพจหรือวัตถุที่ไม่ได้ถูกใช้งาน (ตัวแปรชนิด built-in และอินสแตนซ์) ออกจากหน่วยความจำ ซึ่งเป็นพื้นที่ที่โปรแกรมคอมพิวเตอร์จำเป็นต้องใช้สำหรับการประมวลผล เมื่อพื้นที่ดังกล่าวมีปริมาณมากจะทำให้คอมพิวเตอร์ทำงานได้อย่างมีประสิทธิภาพเพิ่มขึ้น โดย Garbage collection จะพิจารณาพื้นที่หน่วยความจำที่ไม่ได้ถูกใช้งานในเวลานานๆ กลับคืนให้กับระบบ กลไกของ Garbage collection ในไฟรอนจะประมวลผลอยู่ตลอดเวลาขณะที่โปรแกรมกำลังทำงาน และจะทำหน้าที่คืนหน่วยความจำเมื่อถูกกระตุ้น โดยพิจารณาจากตัวแปรที่เรียกว่า ตัวบับการใช้งานของวัตถุ (Object's reference count) เมื่อค่าของตัวบับดังกล่าวเป็น 0 หมายถึงวัตถุดังกล่าวไม่ได้ถูกใช้งานแล้ว Garbage collection จะลบวัตถุนั้นๆ ออกจากหน่วยความจำทันที เมื่อมีการประกาศตัวแปรและวัตถุใหม่ ตัวบับการใช้งานวัตถุจะเพิ่มขึ้นครั้งละ 1 และจะลดลงเมื่อผู้เขียนโปรแกรมใช้คำสั่ง `del` (หรือเรียกอีกชื่อหนึ่งว่า Destructor) ในการลบตัวแปรหรือวัตถุดังกล่าวทิ้งไป หรือถ้ามีการประกาศตัวแปรหรือวัตถุใดๆ ไว้แล้ว แต่ผู้เขียนโปรแกรมไปอ้างอิงตัวแปรใหม่เพิ่มขึ้นอีก โดยไม่ใช้งานตัวแปรหรือวัตถุที่ประกาศไว้แต่เดิม ก็จะเป็นผลให้ Garbage collection ทำงานเช่นเดียวกัน จากรูปที่ 11.8 แสดงตัวอย่างการทำงานของ Garbage collection และตัวบับการใช้งานวัตถุ (Object's reference count)



รูปที่ 11.8 แสดงการทำงานของ Garbage collection และ Object's reference count

จากรูปที่ 11.8 เริ่มต้นเมื่อทำการสร้างวัตถุ `O` โดยกำหนดค่าให้ตัวแปร `X` เท่ากับ 15 แสดงใน (a) ไฟรอนจะค้นหาพื้นหน่วยความจำที่ว่างให้เพื่อเก็บข้อมูลของ `X` และตัวบับวัตถุ (`O object count`) จะมีค่าเท่ากับ 1 เมื่อผู้เขียนโปรแกรมทำการประกาศตัวแปรเพิ่มคือ `Y` และกำหนดค่าให้เท่ากับ `X` และ `Z = Y` ส่งผลให้มีการจดพื้นที่สำหรับเก็บตัวแปร `Y` และ `Z` เพิ่มเติม โดยตัวแปร `X` จะถูกอ้างอิงเพิ่มขึ้น

ดังนั้นค่าในตัวนับวัตถุจะถูกเพิ่มค่าเป็น 3 และแสดงในรูป (b) เมื่อโปรแกรมทำงานไประยะหนึ่ง ค่าของ Y ถูกกำหนดค่าใหม่เท่ากับ 100 และทำการลบตัวแปร Z ทิ้งด้วยคำสั่ง del ส่งผลให้มีการอ้างอิงไปยังตัวแปร X จาก Y และ Z อีก ทำให้ตัวนับวัตถุลดลงจาก 3 เป็น 1 ดังแสดงใน (c) เมื่อโปรแกรมใช้งานไปอีกระยะเวลาหนึ่งตัวแปร X ถูกลบด้วยคำสั่ง del ส่งผลให้ตัวนับวัตถุเป็น 0 ส่งผลกระทบให้ Garbage collection ทำการคืนหน่วยความจำของวัตถุ O ให้กับระบบ ดังรูป (d)

ผู้เขียนโปรแกรมสามารถคืนพื้นที่หน่วยความจำได้เอง เมื่อเห็นเหมาะสมแล้วว่าวัตถุดังกล่าวไม่ได้ถูกใช้งานอีกต่อไป โดยใช้เมธอดชื่อว่า `__del__` (เรียกว่า deconstructor) เมธอดดังกล่าวจะถูกเรียกใช้งานทันทีเมื่อผู้เขียนออกคำสั่ง del กับอินสแตนซ์ของวัตถุใดๆ ที่ต้องการลบ ดังแสดงในโปรแกรมที่ 11.6

Program Example 11.6: destroying the object

```

1 # Destroying any objects that not use
⇒2 class Car:
⇒3     def __init__(self, color='red', speed=100):
4         self.color = color
5         self.speed = speed
6
7     # Destructor method
⇒8     def __del__(self):
8         class_name = self.__class__.name
9         print(class_name, "object has destroyed")
10
11
⇒12 car1 = Car()
⇒13 car2 = car1
⇒14 car3 = car1
15 # prints the ids of the objects
⇒16 print("ID[car1]=", id(car1), ", ID[car2]=", id(car2), ",
16 ID[car3]=", id(car3))
⇒17 del car1
18 del car2
19 del car3

```



ID[car1]= 60680568 , ID[car2]= 60680568 , ID[car3]= 60680568
Car object has destroyed

จากโปรแกรมตัวอย่างที่ 11.6 แสดงการใช้งานเมธอด destructor โดยโปรแกรมเริ่มต้นจากการสร้างคลาส Car ในบรรทัดที่ 2 และประกาศเมธอด constructor ชื่อ `__init__()` ในบรรทัดที่ 3 โดยรับค่าอาร์กิวเมนต์ 2 ตัวคือ `color` และ `speed` (ไม่นับรวม `self`) เมธอดนี้จะทำหน้าที่กำหนดค่าเริ่มต้นต่างๆ เพื่อเตรียมความพร้อมให้กับคลาส Car ได้ทำงาน โดยกำหนดค่าเริ่มต้นของ `color` เท่ากับ 'red' และ `speed` เท่ากับ 100

ในบรรทัดที่ 8 เป็นเมธอด destructor ชื่อว่า `__del__` เมธอดนี้จะถูกเรียกใช้งานเสมอ (เมื่อมีการประกาศไว้ในคลาส) จากคำสั่ง `del` เพื่อทำหน้าที่เก็บข้อมูลหรือทำความสะอาดคลาสก่อนคืนพื้นที่

หน่วยความจำให้กับระบบ สิ่งที่ควรจะทำความสะอาดของเมธอดนี้คือ ตัวแปร และข้อมูลต่างๆ ที่ถูกประกาศไว้ในคลาสนั้นเอง สำหรับในตัวอย่างโปรแกรมนี้จะให้พิมพ์ข้อความว่า “Car object has destroyed” ทุกครั้งเมื่อลบวัตถุ Car ทิ้ง ในบรรทัดที่ 12 เป็นการสร้างอินสแตนซ์จากคลาส Car เป็น car1 หลังจากใช้คำสั่งดังกล่าวเมธอด constructor จะทำงาน ต่อจากนั้นในบรรทัดที่ 13 และ 14 เป็นการกำหนดตำแหน่งที่อยู่ของอินสแตนซ์ car1 ให้กับ car2 และ car3 ตามลำดับ ซึ่งจะส่งผลให้อินสแตนซ์ทั้ง 3 ตัวมีที่อยู่เดียวกันคือ 60680568 (บรรทัดที่ 16) เมื่อทำการลบอินสแตนซ์ car1 (บรรทัดที่ 17) ส่งผลให้เมธอด deconstructor (`__del__()`) ทำงาน โดยแสดงข้อความ “Car object has destroyed” ออกจอภาพ แต่จะแสดงเพียงครั้งเดียวเท่านั้น เพราะว่า car1, car2 และ car3 อ้างอิงวัตถุตัวเดียวกันนั่นเอง

สรุปการทำงานของเมธอด Constructor และ Destructor มีลำดับการทำงานดังนี้คือ

- 1) เมื่อสร้างคลาสจะต้องประกาศเมธอด Constructor (`__init__()`) และ Destructor (`__del__()`) ไว้ด้วย เช่น

class Car:

```
def __init__(self, agr1, arg2,..., argn):
```

กำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ ที่ต้องการใช้งาน

```
def __del__(self):
```

ยกเลิกตัวแปรต่างๆ ภายในคลาสก่อนคืนหน่วยความจำให้ระบบ

- 2) เมธอด Constructor (`__init__()`) จะเริ่มทำงานทันทีหลังจากการสร้างอินสแตนซ์ของวัตถุ เช่น

```
car1 = Car()
```

- 3) เมธอด Destructor (`__del__()`) จะเริ่มทำงานทันทีหลังจากการออกคำสั่ง `del` เพื่อลบอินสแตนซ์ของวัตถุ เช่น

```
del car1
```

การสืบทอดคุณสมบัติของคลาส (Class Inheritance)

การสืบทอดคุณสมบัติของคลาส (Inheritance) เป็นการสืบทอดคุณสมบัติต่างๆ จากคลาสแม่ไปยังคลาสลูก คือ แอ็ตทริบิวต์และเมธอด แนวความคิดที่ดีมากในการสืบทอด คือ คุณสมบัติบางประการที่เหมือนกันควรจะได้รับการสืบทอดไปด้วย ไม่ควรสร้างใหม่ทั้งหมดทุกครั้ง ซึ่งแนวความคิดดังกล่าว สอดคล้องกับความเป็นจริงเกี่ยวกับการดำรงชีวิต เช่น บุตรที่เกิดมาจากการ conjugate และแม่จะได้รับการสืบทอดคุณสมบัติต่างๆ มาด้วย เช่น ตา สีผิว ลักษณะรูปร่างหน้าตา เป็นต้น แต่บุตรสามารถมีลักษณะพิเศษเพิ่มขึ้นได้ เช่น สีผิวอาจจะผสมระหว่างพ่อและแม่ แต่มีความสูงมากกว่าพ่อและแม่ เป็นต้น

สำหรับรูปแบบการสร้างคลาสลูก (Subclass) ในไพธอนจะใช้สัญลักษณ์ (...) ในการอ้างถึงการสืบทอดคุณสมบัติจากคลาสแม่ (Parent class) ดังนี้

```
class SubClassName (ParentClass1[, ParentClass2, ...])
```

ขึ้นต้นด้วยคีย์เวิร์ด class ตามด้วยชื่อคลาสลูกที่ต้องการสร้าง (SubClassName) เปิดวงเล็บแล้วตามด้วยคลาสแม่ที่ต้องการสืบทอดคุณสมบัติ การสืบทอดสามารถสืบทอดมาจากคลาสแม่ได้มากกว่า 1 คลาส (เช่นสืบทอดจากพ่อ + แม่) โดยชื่อคลาสแม่จะอยู่ภายในเครื่องหมาย (...) สำหรับตัวอย่างโปรแกรมแสดงการสืบทอดคุณสมบัติ แสดงในโปรแกรมที่ 11.7

Program Example 11.7: Inheritance of class

```

1   # Showing inheritance, constructor and Destructor
2   # Define parent class
3   class ClassicCar:
4       register_number = 0
5       color = 'white'
6       #Constructor method
7       def __init__(self):
8           ClassicCar.register_number += 1
9           print ("ClassicCar: Constructor")
10
11      def ClassicCarGetReg(self):
12          print ("ClassicCar: Register number is ", self.register_number)
13
14      def ClassicCarSetColor(self, color):
15          self.color = color
16          print ("ClassicCar: Set color")
17
18      def ClassicCarGetColor(self):
19          print ("ClassicCar: Get color is ", ClassicCar.color)
20
21      #Destructor method
22      def __del__(self):
23          class_name = self.__class__.__name__
24          print ("ClassicCar: ", class_name, "object has destroyed")
25
26      # Define Toyota child class
27      class ToyotaCar(ClassicCar):
28          def __init__(self):
29              self.color = 'gold blond'
30              print ("Toyota Car: Constructor")
31
32          def ToyotaCarGetColor(self):
33              print ('Toyota Car: Get color is ', self.color)
34
35      # Define Honda child class
36      class HondaCar(ClassicCar):
37          def __init__(self):
38              self.color = 'black'
```

```

39         print ("Honda Car: Constructor")
40
⇒41     def HondaCarGetColor(self):
42         print ('Honda Car: Get color is ',self.color)
43
44 # Define Dummy child class
⇒45 class DummyCar(ClassicCar):
45     color = 'No color'
⇒46     def DummyCarGetColor(self):
47         print ('Dymmy Car: Get color is ',self.color)
48
49
50 #Creating Toyota car instance
⇒51 car1 = ToyotaCar()
⇒52 car1.ClassicCarGetReg()
⇒53 car1.ClassicCarGetColor()
⇒54 car1.ClassicCarSetColor('green')
⇒55 car1.ClassicCarGetColor()
⇒56 car1.ToyotaCarGetColor()
57 #Creating Honda car instance
⇒58 car2 = HondaCar()
58 car2.ClassicCarGetReg()
59 car2.ClassicCarGetColor()
60 car2.ClassicCarSetColor('yello')
61 car2.ClassicCarGetColor()
62 car2.HondaCarGetColor()
63
64 #Creating Dummay car instance
⇒65 car3 = DummyCar()
65 car3.ClassicCarGetReg()
66 car3.ClassicCarGetColor()
67 car3.ClassicCarSetColor('pink')
68 car3.ClassicCarGetColor()
69 car3.DummyCarGetColor()
70
71 #Destroying all instances
⇒72 del car1
72 del car2
73 del car3

```


OUTPUT

```

Toyota Car: Constructor
ClassicCar: Register number is  0
ClassicCar: Get color is white
ClassicCar: Set color
ClassicCar: Get color is white
Toyota Car: Get color is green

Honda Car: Constructor
ClassicCar: Register number is  0
ClassicCar: Get color is white
ClassicCar: Set color
ClassicCar: Get color is white
Honda Car: Get color is yello

ClassicCar: Constructor
ClassicCar: Register number is  1

```

```

ClassicCar: Get color is white
ClassicCar: Set color
ClassicCar: Get color is white
Dymmy Car: Get color is pink

ClassicCar: ToyotaCar object has
destroyed
ClassicCar: HondaCar object has
destroyed
ClassicCar: DummyCar object has
destroyed

```

จากโปรแกรมที่ 11.7 แสดงตัวอย่างการสืบทอดคุณสมบัติของคลาสรถยนต์ (Classic Car) โดยโปรแกรมเริ่มจากบรรทัดที่ 3 เป็นการสร้างคลาสแม่ชื่อ ClassicCar ซึ่งมีแอ็ตทริบิวต์ 2 ตัวคือ register_number (Class variable) มีค่าเริ่มต้นเป็น 0 และ color (Local variable) มีค่าเท่ากับ “white” บรรทัดที่ 7 สร้างคอนสตรัคเตอร์ที่ทำหน้าที่เพิ่มค่าที่ลະ 1 ให้กับแอ็ตทริบิวต์ register_number ทุกๆ ครั้งเมื่อมีการสร้างอินสแตนซ์ของ ClassicCar บรรทัดที่ 11 ของ ClassicCar สร้างเมธอดชื่อ ClassicCarGetReg() ทำหน้าที่แสดงค่าของ register_number ให้กับผู้เรียกใช้งาน บรรทัดที่ 14 สร้าง เมธอดชื่อ ClassicCarSetColor() ทำหน้าที่กำหนดสีของรถยนต์ ซึ่งต้องการพารามิเตอร์ในการทำงาน 1 ตัวคือ ค่าของสีรถยนต์ บรรทัดที่ 18 สร้างเมธอดชื่อ ClassicCarGetColor() ทำหน้าที่แสดงสีของรถยนต์ และเมธอดสุดท้าย (บรรทัดที่ 22) คือเมธอด Destructor ทำหน้าที่ทำความสะอาดคลาส ในที่นี้ให้สั่ง พิมพ์ว่า คลาสที่ทำงานอยู่ได้ถูกลบไปแล้ว

ในบรรทัดที่ 27 โปรแกรมสร้างคลาสลูกชื่อว่า ToyotaCar ซึ่งได้รับการสืบทอดคุณสมบัติมาจาก คลาส ClassicCar ในบรรทัดถัดมา (บรรทัดที่ 28) คลาส ToyotaCar ทำการสร้าง constructor (`__init__`) ทำหน้าที่กำหนดค่าเริ่มต้นของสีรถโดยตัวเป็นสี “gold blond” และพิมพ์ข้อความว่า “Toyota Car: Constructor” ในคลาส toyotaCar มีการสร้างเมธอดชื่อ ToyotaCarGetColor() ทำหน้าที่แสดงสี ของรถยนต์ (บรรทัดที่ 32)

ในบรรทัดที่ 36 โปรแกรมสร้างคลาสลูกชื่อว่า HondaCar ซึ่งได้รับการสืบทอดคุณสมบัติมาจาก คลาส ClassicCar เหมือนกับคลาส ToyotaCar และสร้าง constructor (บรรทัดที่ 37) ทำหน้าที่ กำหนดค่าเริ่มต้นของสีรถเป็นสี “black” และพิมพ์ข้อความว่า “Honda Car: Constructor” ใน คลาส HondaCar มีการสร้างเมธอดชื่อ HondaCarGetColor() ทำหน้าที่แสดงสีของรถยนต์ (บรรทัดที่ 41)

ในบรรทัดที่ 45 โปรแกรมสร้างคลาสลูกชื่อว่า DummyCar ซึ่งได้รับการสืบทอดคุณสมบัติมา จากคลาส ClassicCar เช่นเดียวกัน แต่ไม่มี constructor (เมื่อคลาส DummyCar ทำงานจะไปเรียกใช้ constructor จากคลาสแม่แทน) บรรทัดที่ 46 ประกาศแอ็ตทริบิวต์ color เพื่อเก็บสีของรถยนต์ โดย

กำหนดค่าเริ่มต้นเป็น “No Color” บรรทัดที่ 47 สร้างเมธอดชื่อ DummyCarGetColor() ทำหน้าที่แสดงสีของรถยนต์

เมื่อทำการประกาศคลาสแม่ (ClassicCar) และคลาสลูกเสร็จแล้ว (ToyotaCar, HondaCar, DummyCar) ขั้นต่อไปโปรแกรมทำการสร้างอินสแตนซ์ของคลาส ToyotaCar เป็น car1 (บรรทัดที่ 51) ส่งผลให้ constructor ในคลาส ToyotaCar ทำงานทันที โดยพิมพ์ข้อความว่า “Toyota Car: Constructor” ในบรรทัดที่ 52 เริ่มเรียกใช้งานเมธอด car1.ClassicCarGetReg() ซึ่งเป็นเมธอดของคลาสแม่ที่ได้รับสืบทอดมา ผลลัพธ์ที่ได้คือ “ClassicCar: Register number is 0” ทั้งนี้ เพราะ constructor ในคลาสแม่ไม่ทำงาน (โปรดจำ: ถ้ามีการสร้าง constructor ไว้ทั้งคลาสแม่และคลาสลูก เมื่อสร้างอินสแตนซ์ constructor ที่คลาสลูกจะถูกเรียกใช้งาน โดยไม่เรียกใช้ constructor ที่คลาสแม่) บรรทัดที่ 53 เรียกเมธอดจากคลาสแม่ชื่อ car1.ClassicCarGetColor() ผลที่ได้คือ “ClassicCar: Get color is white” บรรทัดที่ 54 เรียกเมธอดจากคลาสแม่เพื่อกำหนดสีของรถยนต์ให้เป็นสีเขียว คือ car1.ClassicCarSetColor('green') ผลที่ได้คือ “ClassicCar: Set color” บรรทัดที่ 55 เรียกเมธอดจากคลาสแม่เพื่อแสดงผลของการกำหนดชื่อสีรถยนต์ใหม่จากคำสั่งที่ผ่านมา คือ car1.ClassicCarGetColor() ผลที่ได้คือ “ClassicCar: Get color is white” เมื่อ้อนเดิมทั้งนี้ เพราะว่าแอตทริบิวต์ color เป็นตัวแปรแบบโลคอลของคลาส ซึ่งไม่สามารถเข้าถึงได้จากคลาสลูก (ถ้าต้องการให้เปลี่ยนแปลงค่าได้ต้องเปลี่ยนเป็นตัวแปรชนิด Class variable แทน โดยเปลี่ยนจาก self.color เป็น ClassicCar.color แทน ในบรรทัดที่ 15)

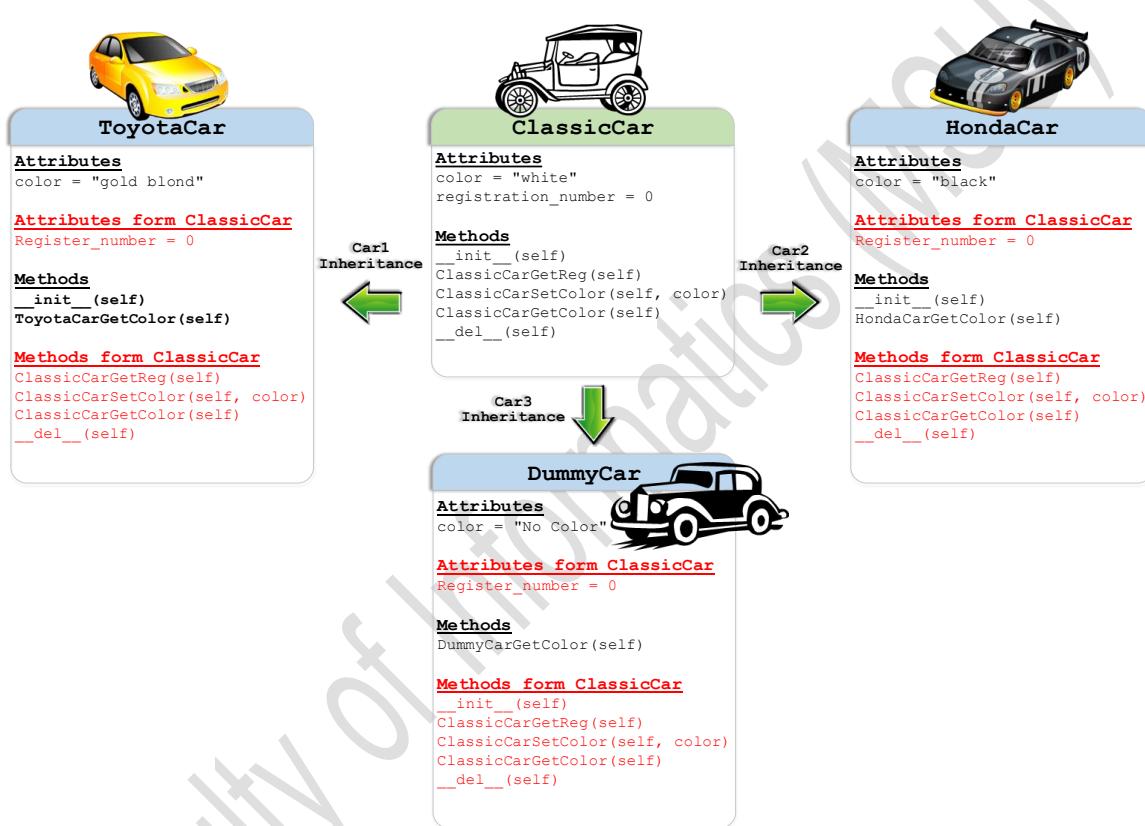
บรรทัดที่ 56 เรียกเมธอดจากคลาส ToyotaCar เอง คือ car1.ToyotaCarGetColor() เพื่อแสดงสีของรถยนต์ ผลที่ได้คือ “Toyota Car: Get color is green” ซึ่งเป็นสีที่กำหนดให้กับคลาสแม่ในบรรทัดที่ 54 แทนที่จะเป็นสี “gold blon” ทั้งนี้ เพราะไฟชอนมองว่า self.color คือแอตทริบิวต์ภายในคลาส ToyotaCar ไม่ใช้คลาส ClassicCar แต่เนื่องจากคลาส ToyotaCar ได้รับการถ่ายทอดคุณสมบัติของเมธอด ClassicCarSetColor() มาด้วย ทำให้สี “green” ที่กำหนดเป็นสี (color) ของคลาสลูกนั่นเอง

สำหรับคลาสลูก HondaCar ก็มีลักษณะการทำงานเหมือนกับ ToyotaCar ทุกประการ ทั้งนี้ เพราะได้รับการถ่ายทอดคุณสมบัติ และมี constructor และ Destructor เมื่อ้อนกันทุกประการ แต่ที่น่าสนใจคือ คลาสลูก DummyCar ไม่มีการสร้าง constructor ไว้ ดังนั้นจึงได้รับเมธอดดังกล่าวถ่ายทอดคุณสมบัติมาจากคลาสแม่ด้วย เมื่อสร้างอินสแตนซ์ของคลาส DummyCar (บรรทัดที่ 65) ส่งผลให้ constructor ที่ได้รับจากคลาสแม่ทำงาน โดยการเพิ่มค่าตัวแปร register_number เป็น 1 เมื่อทำการเรียกเมธอด car3.ClassicCarGetReg() ในบรรทัดที่ 66 ส่งผลให้ได้รับผลลัพธ์คือ “ClassicCar: Register number is 1” เพราะอินสแตนซ์ car3 สามารถเข้าถึงตัวแปรแบบ Class variable “ได้นั่นเอง ในบรรทัดที่ 72 เป็นคำสั่งที่สั่งให้ทำการลบอินสแตนซ์ทั้งหมดออกจากหน่วยความจำ ส่งผลให้คลาสลูกทั้ง 3

อินสแตนซ์เรียกใช้งานเมธอด Destructor (`__del__`) ที่ได้รับการถ่ายทอดจากคลาสแม่ให้มีอนกัน
เนื่องจากไม่มีคลาสลูกใดๆ ที่สร้างเมธอด Destructor ไว้เลย โปรดพิจารณาในรูปที่ 11.9 ประกอบเพื่อ
เพิ่มความเข้าใจ



Note: เมื่อประกาศ constructor (`__init__`) และ Destructor (`__del__`) ไว้ทั้งคลาสแม่ (Parent class) และคลาสลูก (Child class) เมื่อทำการสร้างอินสแตนซ์จะใช้ constructor และ Destructor จากคลาสลูกเป็นหลัก แต่ถ้าคลาสลูกไม่ได้ประกาศไว้จะเรียกจากคลาสแม่มาใช้งานแทน



รูปที่ 11.9 แสดงการสืบทอดคลาสของ ClassicCar, ToyotaCar, HondaCar และ DymmyCar

จากรูปที่ 11.9 แสดงการสืบทอดคลาส โดยมี ClassicCar เป็นคลาสแม่ และคลาส ToyotaCar, HondaCar และ DummyCar เป็นคลาสลูก ผลจากการสืบทอดสรุปได้ดังนี้คือ

ToyotaCar ได้รับการถ่ายทอดคุณสมบัติมาจาก ClassicCar คือ

แอ็ตทริบิวต์	<code>register_number</code>
เมธอด	<code>ClassicCarGetReg(self)</code> , <code>ClassicCarSetColor(self, color)</code> , <code>ClassicCarGetColor(self)</code> และ <code>__del__(self)</code>

HondaCar ได้รับการถ่ายทอดคุณสมบัติมาจาก ClassicCar คือ

แอตทริบิวต์	register_number
เมธอด	ClassicCarGetReg(self), ClassicCarSetColor(self, color), ClassicCarGetColor(self) และ __del__(self)

DummyCar ได้รับการถ่ายทอดคุณสมบัติมาจาก ClassicCar คือ

แอตทริบิวต์	register_number
เมธอด	__init__(self), ClassicCarGetReg(self), ClassicCarSetColor(self, color), ClassicCarGetColor(self) และ __del__(self)

เมื่อผู้เขียนโปรแกรมต้องการสืบทอดคุณสมบัติจากคลาสแม่มากกว่า 1 คลาส มีรูปแบบคำสั่งดังนี้

```
class ClassicCar:      # define Classic Car class
    statement(s)

class GeneralCar:     # define General Car calss
    statement(s)

class ToyotaCar (ClassicCar, GeneralCar): # subclass of ClassicCar and GeneralCar
    statement(s)
```

จากตัวอย่างด้านบนเป็นตัวอย่างของสืบทอดคุณสมบัติของคลาสแม่มากกว่า 1 คลาส โดยคลาส ToyotaCar จะสืบทอดคุณสมบัติมาจากทั้งคลาส ClassicCar และ GeneralCar โดยประกาศคลาสแม่ที่ต้องการสืบทอดไว้ในเครื่องหมายวงเล็บ (...) ไฟชอนไม่ได้แนะนำไว้ว่าควรจะสืบทอดคุณสมบัติจากคลาสแม่ได้มากแค่ไหน ขึ้นอยู่กับความเหมาะสมและความซับซ้อนของบัญหาแต่ละบัญหา

ไฟชอนได้เตรียมฟังชันชื่อ issubclass(sub, sup) เอาไว้ให้นักเขียนโปรแกรมใช้เพื่อตรวจสอบว่า sub (คลาสลูก: subclass) สืบทอดมาจาก sup (คลาスマ่: super class) หรือไม่ เป็นจริงเมื่อทั้งสองคลาสมีความสัมพันธ์กัน เช่น

issubclass(ToyotaCar, ClassicCar) \Rightarrow True

issubclass(ToyotaCar, DummyCar) \Rightarrow False

ฟังชันชื่อ isinstance(obj, class) ให้ผลลัพธ์เป็นจริงเมื่อ obj (Object) เป็นอินสแตนซ์ของ class เช่น

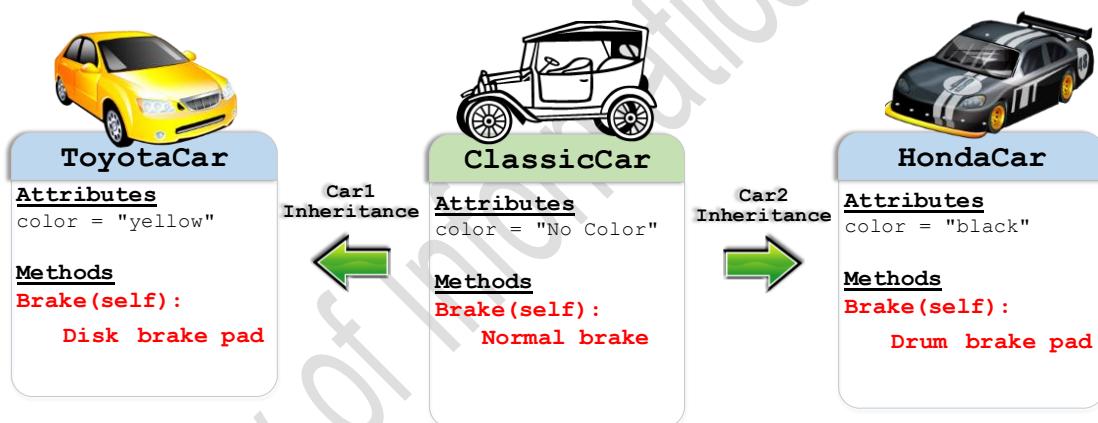
isinstance(car1, ClassicCar) \Rightarrow True

isinstance(ToyotaCar, ClassicCar) \Rightarrow False

การโอเวอร์ไรร์ดิ้งเมธอด (Overriding methods)

แนวความคิดการโอเวอร์ไรร์ดิ้งเกิดขึ้นจาก คุณสมบัติของพุติกรรมที่ได้รับการสืบทอดจากคลาสแม่ไปให้คลาสลูกไม่เหมาะสมที่จะใช้งานภายในคลาสลูก ดังนั้นคลาสลูกสามารถแก้ไขคุณสมบัติที่รับสืบทอดเหล่านั้นได้ ตัวอย่างเช่น คุณพ่อและคุณแม่ทำงานในเวลาปกติ คือจันทร์-ศุกร์ (หยุดวันเสาร์และอาทิตย์) ลูกชายก็ทำงานเหมือนกัน (ได้รับการถ่ายทอดเรื่องการทำงานมา) แต่ลูกชายทำงานในลักษณะการให้บริการ ดังนั้นจึงไม่มีเวลาทำงานที่แน่นอนและไม่มีวันหยุด (เป็นการทำงานเหมือนกันแต่มีรายละเอียดในการทำงานที่ต่างกัน เรียกว่า โอเวอร์ไรร์ดิ้ง)

สำหรับการเขียนโปรแกรมเชิงวัตถุ เป็นการสร้างเมธอดของคลาสลูกที่มีชื่อเหมือนกันกับเมธอดในคลาสแม่ แต่ทำงานแตกต่างจากเมธอดในคลาสแม่นั้นเอง เพื่อประกอบความเข้าใจเพิ่มเติมโปรดพิจารณารูปที่ 11.10 ประกอบเพิ่มเติม



รูปที่ 11.10 แสดงการโอเวอร์ไรร์ดิ้งเมธอดสำหรับคลาสรถยนต์

จากรูปที่ 11.10 แสดงให้เห็นว่าคลาสลูก (ToyotaCar และ HondaCar) ได้รับการถ่ายทอดคุณสมบัติการเบรค (Brake) แบบธรรมดามาจากคลาสแม่คือ ClassicCar แต่ว่าเมื่อนำมาใช้งานแล้วกลับเกิดความไม่เหมาะสม (เช่น การเบรคแบบเดิมไม่มีประสิทธิภาพเกิดอุบัติเหตุบ่อยครั้ง) ดังนั้นรถยนต์ Toyota จึงดัดแปลงคุณภาพของเบรคใหม่ให้กลายเป็นระบบแบบ Disk brake pad (เรียกชื่อการเบรคเหมือนกันแต่วิธีการทำงานของเบรคต่างกัน) ส่วน Honda ก็ปรับปรุงระบบเบรคของตนเองใหม่เพื่อให้เหมาะสมกับสภาพปัจจุบันเช่นกัน โดยมีลักษณะรายละเอียดที่ต่างกัน เรียกว่า Drum brake pad เป็นต้น

เพื่อให้เห็นภาพของการโอเวอร์ไรร์ดิ้งเมธอดในทางการเขียนโปรแกรม โปรดพิจารณาตัวอย่างโปรแกรมที่ 11.8 โดยใช้ตัวอย่างของคลาสรถยนต์ที่ได้อธิบายไปแล้วข้างต้น

Program Example 11.8: Overriding methods

```

1   # Showing overriding methods
2   # Define Classic car parent class
3   class ClassicCar:
4       #ClassicCar method
5       def brake(self):
6           print ("ClassicCar: Normal brake!")
7
8   # Define Toyota car child class
9   class ToyotaCar(ClassicCar):
10      #ToyotaCar method
11      def brake(self):
12          print ("ToyotaCar: Disk brake pad!")
13
14  # Define Honda car child class
15  class HondaCar(ClassicCar):
16      #HondaCar method
17      def brake(self):
18          print ("HondaCar: Drum brake pad!")
19
20  # Define Dummy car child class
21  class DummyCar(ClassicCar):
22      pass
23
24  #Creating Toyota car instance
25  car1 = ToyotaCar()
26  car1.brake()
27  #Creating Honda car instance
28  car2 = HondaCar()
29  car2.brake()
30  #Creating Dummay car instance
31  car3 = DummyCar()
32  car3.brake()

```



ToyotaCar: Disk brake pad!
HondaCar: Drum brake pad!
ClassicCar: Normal brake!

จากโปรแกรมที่ 11.8 เป็นตัวอย่างการทำงานของโอเวอร์ไรร์ดิ้งเมธอด บรรทัดที่ 3 สร้างคลาสแมชชีน ClassicCar โดยมีเมธอดเดียวกับ brake (บรรทัดที่ 5) ในตัวอย่างนี้ สมมติว่าเป็นระบบเบรคแบบธรรมด้า (Normal brake) บรรทัดที่ 9 สร้างคลาสลูกชิ้น ToyotaCar ที่ได้รับการสืบทอดคุณสมบัติมาจากการ ClassicCar และทำโอเวอร์ไรร์ดิ้งเมธอด brake ไว้ด้วย (ชิ้นเมธอดเป็นชิ้นเดียวกันกับชิ้นเมธอดในคลาสแมชชีน) โดยเปลี่ยนระบบเบรคเป็นแบบ Disk brake pad (บรรทัดที่ 11) สำหรับคลาลูก HondaCar ก็ทำในลักษณะเดียวกับ ToyotaCar คือ ทำโอเวอร์ไรร์ดิ้งเมธอด brake เมื่อกันกัน แต่เปลี่ยนเทคนิคการเบรคคือ Drum brake pad (บรรทัดที่ 15 และ 17) และสร้างคลาส DummyCar (บรรทัดที่ 21) เป็นคลาสลูกที่สืบทอดคุณสมบัติมาจาก ClassicCar เช่นกัน แต่ไม่มีการสร้างเมธอดใดๆ ไว้เลย คำสั่ง pass ในบรรทัดที่ 22 เป็นคำสั่งที่รักษาโครงสร้างของไวยกรณ์ไว้ไม่ให้ผิดพลาด ในตัวอย่างนี้ถ้าไม่ใส่คำสั่ง

pass จะเกิด excepted an intented block เพราะไฟชอนมองว่า คำสั่ง car1 = ToyotaCar() ในบรรทัดที่ 25 เป็นคำสั่งของคลาส DummyCar แต่มีอยู่หน้าไม่ตรงกับคลาส DummyCar นั้นเอง

ผลจากการสั่งรันโปรแกรม ปรากฏว่าเมธอด brake ในอินสแตนซ์ car1 (ToyotaCar) พิมพ์ข้อความว่า “ToyotaCar: Disk brake pad!” ส่วนเมธอด brake ในอินสแตนซ์ car2 (HondaCar) พิมพ์ข้อความว่า “HondaCar: Drum brake pad!” ทั้งนี้ เพราะทั้งสองเมธอดได้ถูกทำโดยอิร์เรด์ดิ้งเมธอดไว้นั้นเอง ทำให้การทำงานต่างไปจากคลาสแม่ แต่ในอินสแตนซ์ของคลาส car3 (DummyCar) ไม่มีเมธอดที่โดยอิร์เรด์ดิ้งไว้ ดังนั้นจึงเรียกใช้เมธอดจากคลาสแม่แทน ผลลัพธ์คือพิมพ์ข้อความ “ClassicCar: Normal brake!”

รายชื่อเมธอดในตารางต่อไปนี้เป็นเมธอดของไฟชอนที่ผู้เขียนโปรแกรมสามารถทำโดยอิร์เรด์ดิ้งได้

เมธอด	คำอธิบาย
<code>__init__(self [,args...])</code>	เมธอด constructor ทำหน้าที่กำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ ในคลาส ก่อนเริ่มทำงาน โดยตัวแปร args จะมีหรือไม่มีก็ได้ เช่น <pre>class ClassicCar: def __init__(self, message): print (message) class ToyotaCar(ClassicCar): pass car1 = ToyotaCar ("Toyota car") ผลลัพธ์คือ Toyota car</pre>
<code>__del__(self)</code>	เมธอด Destructor ทำหน้าที่เครียร์ข้อมูลต่างๆ ก่อนคืนหน่วยความจำ <pre>class ClassicCar: def __del__(self): print ("Classic Car has destroyed!") class ToyotaCar(ClassicCar): def __del__(self): print ("Toyota Car has destroyed!") car1 = ToyotaCar() del car1 ผลลัพธ์คือ Toyota Car has destroyed!</pre>
<code>__repr__(self)</code>	เมธอดแสดงผลของสตริง เรียกใช้โดย <code>repr(obj)</code>
<code>__str__(self)</code>	เมธอดแสดงผลของสตริง เรียกใช้โดย <code>str(obj)</code>
<code>__cmp__(self, x)</code>	เมธอดที่ใช้ในการเปรียบเทียบ เรียกใช้โดย <code>cmp(obj, x)</code>

การโอเวอร์โหลดดึงเมธอด (Overloading method/function)

การโอเวอร์โหลด (Overloading) มีลักษณะคล้ายคลึงกับโอเวอร์ไรร์ด (Overriding) คือ ชื่อของ เมธอดต้องมีชื่อเหมือนกัน แตกต่างกันที่การโอเวอร์โหลดจะมีพารามิเตอร์ที่มีจำนวนไม่เท่ากัน ส่วน โอเวอร์ไรร์ดจะต้องมีจำนวนพารามิเตอร์เท่ากันและรูปแบบเดียวกัน โอเวอร์โหลดอาจจะอยู่ในคลาสเดียวกันหรือไม่ก็ได้ แต่โอเวอร์ไรร์ดจะต้องมีการสืบทอดจากคลาสแม่สู่คลาสลูก แสดงในตัวอย่าง โปรแกรมที่ 11.9

Program Example 11.9: overloading methods

```

1   # Overloading methods
2   class Car:
3       #Constructor overloading
4       def __init__(self, *args):
5           if(len(args)==0):
6               print("No overloading")
7           else:
8               print("There are %d arguments overloading"
9 % (len(args)))
10          lists = []
11          for i in args:
12              lists.append(i)
13          print("Constructor overloading arguments:",lists)
14
15      #Method overloading
16      def printX(*args):
17          if(len(args)==0):
18              print("No overloading")
19          else:
20              print("There are %d arguments overloading"
21 % (len(args)-1))
22              lists = []
23              for i in args:
24                  lists.append(i)
25              print("Method overloading arguments:",lists[1:])
26
27      #Creating instance
28      car1 = Car()
29      car2 = Car("Toyota")
30      Car3 = Car("Toyota", "Honda", "BMW", "Audi", "Ford")
31      car1.printX()
32      car1.printX(5)
33      car1.printX(5, 6.5, -5, "Overloading")

```



OUTPUT

```

No overloading
There are 1 arguments overloading
Constructor overloading arguments: ['Toyota']
There are 5 arguments overloading

```

```
Constructor overloading arguments: ['Toyota', 'Honda',
'BMW', 'Audi', 'Ford']
```

```
There are 0 arguments overloading
```

```
Method overloading arguments: []
```

```
There are 1 arguments overloading
```

```
Method overloading arguments: [5]
```

```
There are 4 arguments overloading
```

```
Method overloading arguments: [5, 6.5, -5, 'Overloading']
```

จากตัวอย่างโปรแกรมที่ 11.9 แสดงการสร้างเมธอดโอลิจาร์โลลดิ้ง บรรทัดที่ 4 สร้างคลาส Car

และมีเมธอด constructor (`__init__`) เป็นชนิดโอลิจาร์โลลดิ้ง โดยมีพารามิเตอร์ชนิดทัพเพิล (*args) แบบไม่จำกัดจำนวน บรรทัดที่ 5 โปรแกรมทำการตรวจสอบว่าความยาวของพารามิเตอร์ดังกล่าวบรรจุค่าได้ๆ มาหรือไม่ (`len(args) == 0`) ถ้าไม่มีมากับพารามิเตอร์ดังกล่าวจะพิมพ์ข้อความ “No overloading” แต่ถ้าความยาวไม่เท่ากับ 0 จะทำงานหลัง else ในบรรทัดที่ 7 โดยการพิมพ์ข้อความว่า “There are X arguments overloading” ในบรรทัดที่ 8 โดย X คือ จำนวนอาร์กิวเมนต์ที่รับเข้ามา ในบรรทัดที่ 9 โปรแกรมทำการดึงข้อมูลออกจากอาร์กิวเมนต์ args (ในบรรทัดที่ 9-12) เก็บไว้ในตัวแปรลิสต์ชื่อ lists โดยใช้ for ดึงค่ามาทีละค่าเก็บไว้ในตัวแปร i ซึ่งข้อมูลที่นำออกจากรากิวเมนต์ args ในแต่ละรอบจะเก็บแบบต่อเนื่องไปเรื่อยๆ (append) เมื่อได้ข้อมูลครบถ้วนแล้ว ทำการส่งพิมพ์ออกจากภาพ

บรรทัดที่ 15 โปรแกรมสร้างเมธอดโอลิจาร์โลลดิ้งชื่อ `printX` ซึ่งสามารถรับพารามิเตอร์ได้ไม่จำกัด การทำงานคล้ายกับ Constructor overloading ที่ได้กล่าวมาแล้ว แต่แตกต่างกันตรงที่เมธอดนี้ไม่มีการรับค่าพารามิเตอร์ self เข้ามาทำงานด้วย แต่เพื่อนจะแบบใส่มาให้โดยที่ผู้เขียนโปรแกรมไม่ทราบ เมื่อทำการพิมพ์ค่ามาแสดง ในบรรทัดที่ 23 ส่งผลให้โปรแกรมพิมพ์ค่าที่อยู่ของวัตถุติดมาด้วย คือ [`<__main__.Car object at 0x0000000038279B0>`] ดังนั้นเพื่อผลลัพธ์ที่ถูกต้อง ให้เลื่อนพิมพ์ข้อมูลใน lists ออกไปอีก 1 ตำแหน่งด้วยคำสั่ง `lists[1:]` จะทำให้ได้ข้อมูลที่ถูกต้อง

เมื่อโปรแกรมสร้างอินสแตนซ์ car1, car2 และ car3 ของคลาส Car ในบรรทัดที่ 25, 26 และ 27 จะทำให้ constructor ที่ทำโอลิจาร์โลลดิ้งทำงาน สังเกตว่า การสร้างอินสแตนซ์ในแต่ละครั้งจะส่งอาร์กิวเมนต์ไม่เท่ากัน แต่ constructor ก็สามารถทำงานได้อย่างถูกต้อง ในบรรทัดที่ 28, 29 และ 30 ทดสอบเรียกเมธอดที่ทำโอลิจาร์โลลดิ้ง ชื่อว่า `printX()` และส่งอาร์กิวเมนต์ในจำนวนที่ต่างๆ กันให้กับเมธอดดังกล่าว `printX()` สามารถแสดงผลได้อย่างถูกต้องเช่นกัน

หมายเหตุ: การโอลิจาร์โลลดิ้งเมธอดในไฟรอนไม่เหมือนกับใน C++ หรือ Java โดย C++ มีรูปแบบดังนี้

```
void myfunction (arg1) { //some code }
```

```
void myfunction (arg1, arg2) { //some code }
```

```
void myfunction (arg1, arg2, arg3,...,argn) { //some code }
```

ถ้าต้องการโอเวอร์โหลด 10 พารามิเตอร์จะต้องสร้างอย่างน้อย 10 เมธอด และการรับส่งพารามิเตอร์ต้องมีชนิดของตัวแปรที่ตรงกันทั้งหมด มิเช่นนั้นจะเกิดข้อผิดพลาดขึ้น

แต่ Python ใช้เพียงเมธอดและอาร์กิวเมนต์เดียวคือ *args ซึ่งเป็นตัวแปรชนิดทัพเพลิ

```
def myfunction (*args): some code
```



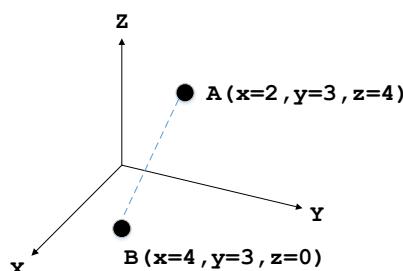
Caution! Overriding method คือ เมธอดที่สืบทอดมาจากคลาสแม่ แต่มีคุณสมบัติไม่ตรงกับการใช้งาน คลาสลูกจึงทำการเปลี่ยนแปลงรายละเอียดการทำงานข้างในเมธอดใหม่ แต่ชื่อและอาร์กิวเมนต์ยังคงเหมือนเดิม, Overloading method คือ ต้องการใช้เมธอดมีความสามารถรับพารามิเตอร์ได้ไม่จำกัด โดยมีชื่อเหมือนเดิม แต่พารามิเตอร์ไม่จำกัด และไม่จำเป็นต้องอยู่ในคลาสเดียวกัน

การโอเวอร์โหลดตัวดำเนินการ (Overloading operators)

การโอเวอร์โหลดตัวดำเนินการ หมายถึง การอนุญาตให้วัตถุที่สร้างขึ้นจากคลาสใดๆ สามารถใช้เครื่องหมายตัวดำเนินการ (Operators) พื้นฐานได้ เช่น เครื่องหมาย บวก ลบ คูณ หาร (+, -, *, /) และอื่นๆ จากหัวข้อที่ผ่านมาเมธอดของ Python ที่ยอมให้โอเวอร์โหลดได้คือ __init__, __del__, __repr__, __str__ และ __com__ และยังมีเมธอดอื่นๆ อีกมากมายที่สามารถทำโอเวอร์โหลดได้ เช่น __lt__, __le__, __gt__ อ่านเพิ่มเติมได้จาก <http://www.rafekettler.com/magicmethods.html>

สำหรับตัวดำเนินการ (Operator) ที่ภาษา Python อนุญาตให้โอเวอร์โหลดได้ เช่น __add__(), __sub__, __mul__(), __div__() และอื่นๆ อีกจำนวนมากอ่านเพิ่มเติมได้จาก <http://docs.python.org/2/library/operator.html>

ตัวอย่างการโอเวอร์โหลดตัวดำเนินการแสดงในโปรแกรมที่ 11.10 ซึ่งเป็นตัวอย่างการประยุกต์ใช้การโอเวอร์โหลดตัวดำเนินการกับการหาระยะทางจากจุดสองจุดใดๆ ในปริภูมิ 3 มิติ ดังรูปที่ 11.11



รูปที่ 11.11 แสดงจุดของ A และ B ในปริภูมิ 3 มิติ

จากรูปแสดงจุด 2 จุด คือ จุด A มีค่าแกน $x = 2$, แกน $y = 3$ และ แกน $z = 4$ และจุด B มีค่า $x = 4$, $y = 3$ และ $z = 0$ ตามลำดับ ให้หาระยะทางจากจุด A ไปจุด B ซึ่งมีสมการดังต่อไปนี้

$$\begin{aligned} \text{ระยะทางจาก } A \Rightarrow B &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad \dots\dots (1) \\ &= \sqrt{(4 - 2)^2 + (3 - 3)^2 + (0 - 4)^2} \\ &= 4.47 \end{aligned}$$

Program Example 11.10: calculating distance between A and B point

```

1   # Overloading operators
2   import math
3   class Vector:
4       x = y = z = 0
5   ↪3   def __init__(self, x, y, z):
6       self.x = x
7       self.y = y
8       self.z = z
9
10    # Overloading function print()
11   ↪5   def __str__(self):
12       return '-->(' + str(self.x) + ',' + str(self.y) + ',' +
13           str(self.z) + ')'
14
15    # Overloading operator +
16   ↪11   def __add__(self, obj):
17       return Vector(self.x + obj.x, self.y + obj.y, self.z +
18           obj.z)
19
20    # Overloading operator -
21   ↪19   def __sub__(self, obj):
22       return Vector(self.x - obj.x, self.y - obj.y, self.z -
23           obj.z)
24
25    # Overloading operator **
26   ↪23   def __pow__(self, obj):
27       return Vector(self.x ** obj.x, self.y ** obj.y, self.z
28           ** obj.z)
29
30    def squareRoot(self):
31       return math.sqrt(self.x + self.y + self.z)
32
33    A = Vector(2, 3, 4)
34    B = Vector(4, 3, 0)
35    C = B - A
36    print (C)
37    D = Vector(2, 2, 2)
38    E = C ** D
39    print (E)
40    Y = E.squareRoot()
41    print (Y)
42
43    ↪26
44    ↪29
45    ↪37

```



(2, 0, -4)
(4, 0, 16)
4.47213595499958

จากตัวอย่างโปรแกรมที่ 11.10 บรรทัดที่ 3 สร้างคลาสชื่อ Vector มีตัวแปร x, y และ z โดยมีค่าเริ่มต้นเป็น 0 บรรทัดที่ 5 เป็น constructor ที่กำหนดค่า x, y และ z เมื่อโปรแกรมมีการสร้างอินสแตนซ์ บรรทัดที่ 11 เป็นการสร้างโอลเวอร์โหลดเมธอด print() มาทำงาน ซึ่งก็คือเมธอด `__str__(self)` เมื่อโปรแกรมเรียกคำสั่ง `print(Object)` เมื่อใด ไฟรอนจะค้นหาคำสั่ง `print` ในคลาสก่อนถ้าไม่พบจึงเรียก `print` จากไลบรารีของไฟรอนต่อไป แต่ผลลัพธ์ที่ได้จากคำสั่ง `print` ของไลบรารีในไฟรอนจะไม่ถูกต้อง โดยพิมพ์เป็นที่อยู่อุปกรณ์แทน เช่น `<__main__.Vector object at 0x00000000042AE9B0>` ผลลัพธ์ที่ถูกต้องจากคำสั่ง `print` คือ โปรแกรมจะพิมพ์ --> (ค่าของ x, ค่าของ y, ค่าของ z) ตามลำดับ

บรรทัดที่ 15 เป็นคำสั่งที่ใช้สำหรับโอลเวอร์โหลดตัวดำเนินการบวก (+) โดย `obj` คือวัตถุที่ส่งเข้ามาในเมธอด แต่ละวัตถุจะประกอบด้วยตัวแปร x, y และ z เมื่อร่วมค่าในดำเนินการบวกของ x, y และ z แล้ว จะส่งค่ากลับเป็น Vector บรรทัดที่ 19 เป็นคำสั่งที่ใช้สำหรับโอลเวอร์โหลดตัวดำเนินการลบ (-) บรรทัดที่ 23 เป็นคำสั่งที่ใช้สำหรับโอลเวอร์โหลดตัวดำเนินการยกกำลัง (`pow` หรือ `**`) และบรรทัดที่ 26 เป็นการทำค่ารากที่สองของค่า x, y และ z ใน Vector

ผลการสร้างอินสแตนซ์ในบรรทัดที่ 29 ส่งผลให้ A เป็นวัตถุชนิด Vector เก็บข้อมูล 3 ค่าคือ x = 2, y = 3 และ z = 4 ในบรรทัดที่ 30 วัตถุ B มีค่า x = 4, y = 3 และ z = 0 บรรทัดที่ 31 ทำการลบค่าในวัตถุ A และ B โดยโอลเวอร์โหลดตัวดำเนินการลบ ค่าที่ลบกันจะเกิดจาก x ของวัตถุ B ลบกับ x ของวัตถุ A คูณร่วมกัน ($B.x - A.x, B.y - A.y, B.z - A.z$) ตามลำดับ ดังแสดงในสมการที่ (1) ผลที่ได้เก็บไว้ในวัตถุ C บรรทัดที่ 32 ทำการพิมพ์ค่าในวัตถุ C ผลลัพธ์ที่ได้คือ $\Rightarrow (2, 0, -4)$

บรรทัดที่ 33 เป็นการสร้างวัตถุชนิด Vector ให้กับวัตถุ D เพื่อเอาไว้สำหรับเป็นเลขยกกำลังสอง บรรทัดที่ 34 ทำการยกกำลัง (ใช้โอลเวอร์โหลดตัวดำเนินการ `**`) โดยมีค่าของตัวแปรในวัตถุ C เป็นฐาน และค่าของตัวแปรในวัตถุ D เป็นเลขยกกำลัง คือ $C.x^{D.x}, C.y^{D.y}, C.z^{D.z} = 2^2, 0^2, -4^2$ เมื่อสั่งพิมพ์ได้ผลลัพธ์ด้วยโอลเวอร์โหลดเมธอด `print` (บรรทัดที่ 35) ได้ผลลัพธ์เท่ากับ $\Rightarrow (4,0,16)$ บรรทัดที่ 36 และ 37 เรียกฟังชัน `squareRoot()` เพื่อหาค่ารากที่สองของ $E.x + E.y + E.z$ และพิมพ์ผลลัพธ์ออกจอภาพ ได้ค่าตอบคือ 4.472



Note: การโอลเวอร์โหลดตัวดำเนินการจะกระทำได้ก็ต่อเมื่อ ตัวดำเนินการเหล่านี้ไฟรอนอนัญญาตให้ใช้งานได้เท่านั้น และผู้เขียนโปรแกรมต้องจำให้ได้ว่า เมธอดใดคูณกับตัวดำเนินการใด เช่น ตัวดำเนินการ + จะคูณกับ `__add__`, `**` คูณกับ `__pow__`, `*` คูณกับ `__mul__` เป็นต้น ดูได้จาก <http://docs.python.org/2/reference/datamodel.html>

การห่อหุ้มข้อมูล/การซ่อนข้อมูล (Encapsulation/Data hiding)

การห่อหุ้มหรือการซ่อนข้อมูล เป็นกลไกอีกอย่างหนึ่งที่สำคัญในการออกแบบโปรแกรมเชิงวัตถุ โดยมีเป้าหมายเพื่อต้องการรักษาความปลอดภัยหรือป้องกันการเข้าถึงข้อมูลจากภายนอกคลาสในภาษาไพธอนสามารถทำได้โดยใส่เครื่องหมาย __ (2 underscore) ไว้ที่ด้านหน้าตัวแปร เช่น __color สำหรับตัวอย่างการห่อหุ้มข้อมูลพิจารณาในโปรแกรมที่ 11.11

Program Example 11.11: Data hiding

```

1   # Encapsulation/Data hiding
2   class Car:
3       __color = 'red'
4       register_count = 0
5
6       def getColor(self):
7           print("Color is:",self.__color)
8
9       def setColor(self, color):
10      self.__color = color
11      print("Color is:",self.__color)
12
13      def regCount(self):
14          self.register_count += 1
15          print("Register count is:",self.register_count)
16
17      car1 = Car()
18      car1.getColor()
19      car1.setColor("yellow")
20      car1.regCount()
21      car1.__color = 'green'
22      car1.getColor()
23      car1.register_count = 5
24      print(car1.register_count)
25      car1.__Car_color = 'pink'
26      car1.getColor()

```



OUTPUT
 Color is: red
 Color is: yellow
 Register count is: 1
 Color is: yellow
 5
 Color is: pink

จากตัวอย่างโปรแกรมที่ 11.11 แสดงการเขียนโปรแกรมตามแนวคิดการห่อหุ้มข้อมูล เริ่มจากในบรรทัดที่ 2 เป็นการสร้างคลาส Car ที่มีแอ็ตทริบิวต์ 2 ตัวคือ __color = 'red' ซึ่งเป็นแอ็ตทริบิวต์ที่ได้รับการปกป้องหรือการห่อหุ้มไม่ให้สามารถเข้าถึงจากภายนอกคลาสได้ และแอ็ตทริบิวต์ register_count = 0 ซึ่งเป็นแอ็ตทริบิวต์ธรรมดานี้สามารถเข้าถึงจากภายนอกคลาสได้ บรรทัดที่ 6 คือ เมื่อต้องgetColor(self) ทำหน้าที่แสดงสีของรถยนต์ปัจจุบัน บรรทัดที่ 9 คือเมื่อต้องsetColor(self, color)

ซึ่งมีพารามิเตอร์ 1 ตัวคือ สีที่ต้องการกำหนดให้กับรถยนต์ พร้อมกับแสดงสีที่กำหนดใหม่ออกจากภาพ
จากภาพ บรรทัดที่ 13 คือเมธอด regCount() ทำหน้าที่เพิ่มค่า register_count ครั้งละ 1 เมื่อมีการเรียก
เนื้อหาดังกล่าว พร้อมกับพิมพ์ค่าของ register_count ปัจจุบันออกจากภาพ

บรรทัดที่ 17 โปรแกรมสร้างอินสแตนซ์ car1 จากคลาส Car ในบรรทัดที่ 18 โปรแกรมเรียก
getColor() ผลลัพธ์ที่ได้คือ “red” จากนั้นบรรทัดที่ 19 โปรแกรมทำการกำหนดสีใหม่ให้รถยนต์เป็นสี
เหลือง setColor(“yellow”) และพิมพ์สีที่กำหนดออกจากภาพ ผลลัพธ์ที่ได้คือ “Color is: yellow”
บรรทัดที่ 20 โปรแกรมเรียกเมธอด regCount() ผลที่ได้คือค่า register_count เพิ่มค่าเป็น 1 ต่อจากนั้น
บรรทัดที่ 21 โปรแกรมทดลองกำหนดค่าให้กับแอตทริบิวต์ __color โดยตรง ผ่านทางชื่ออินสแตนซ์
จากนั้นบรรทัดที่ 22 เรียกเมธอด getColor() เพื่อแสดงผลที่เกิดจากคำสั่งในบรรทัดที่ 21 ผลปรากฏว่า
สีของรถเป็นสี “yellow” เมื่อนเดิม ทั้งนี้เป็นเพราะแอตทริบิวต์ __color ได้รับการปักป้องการเข้าถึงจาก
ภายนอกตรงๆ การปรับปรุงแอตทริบิวต์ __color จะต้องกระทำการผ่านเมธอดที่โปรแกรมเตรียมไว้ให้
เท่านั้น บรรทัดที่ 23 ทดสอบกำหนดค่าของ register_count ผ่านอินสแตนซ์ตรงๆ เมื่อนในบรรทัดที่
21 ให้มีค่าเท่ากับ 5 และทดสอบพิมพ์ข้อมูลโดยใช้คำสั่ง print(car1.register_count) ในบรรทัดที่ 24 ผล
ปรากฏว่า ค่าในแอตทริบิวต์ register_count ถูกกำหนดใหม่เป็น 5

แม้ว่า Python จะออกแบบให้มีการปักป้องข้อมูลตามแนวความคิดของการโปรแกรมวัตถุแล้วก็
ตาม แต่ Python มีข้อยกเว้นในการเข้าถึงแอตทริบิวต์จากภายนอกไว้ เช่น กัน คือ ผู้เขียนโปรแกรม
สามารถเข้าถึงแอตทริบิวต์จากภายนอกคลาสโดยใช้รูปแบบดังนี้

Instance of Object._className__Attribute

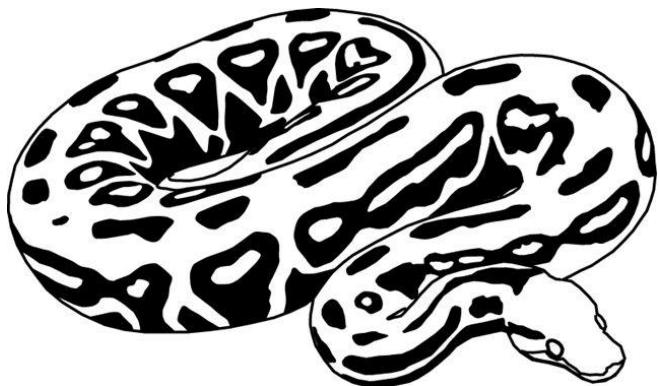
จากตัวอย่างโปรแกรมในบรรทัดที่ 25 คือ car1._Car__color = ‘pink’ บรรทัดที่ 26 ทดสอบพิมพ์สีของ
รถยนต์ใหม่อีกรอบ ผลลัพธ์ที่ได้คือ “Color is: pink”



Note: การห่อหุ้มข้อมูลหรือการซ่อนข้อมูลจากการเรียกใช้จากภายนอกคลาสทำได้โดยใช้
รูปแบบคือ __attribute (2 underscore) แต่ Python มีข้อยกเว้นการเข้าถึงแอตทริบิวต์ดังกล่าว
ได้โดยใช้คำสั่ง InstanceOfClass._ClassName__Attribute เช่น car1._Car__color = “pink”

ฉบับที่ 2

Faculty of Informatics (MSU)



ภาคที่ 3

การเขียนโปรแกรมภาษาไพธอนขั้นสูง



(Advanced

Language Programming)

บทที่ 12 นิพจน์ปกติ และการประมวลผลข้อความ

บทที่ 13 ไฟชอนกับฐานข้อมูล

บทที่ 14 เว็บและซีจีไอ

บทที่ 15 การเขียนโปรแกรมกับมัลติเครดและระบบเครือข่าย

บทที่ 16 การเขียนโปรแกรมระบบสารสนเทศภูมิศาสตร์เบื้องต้น

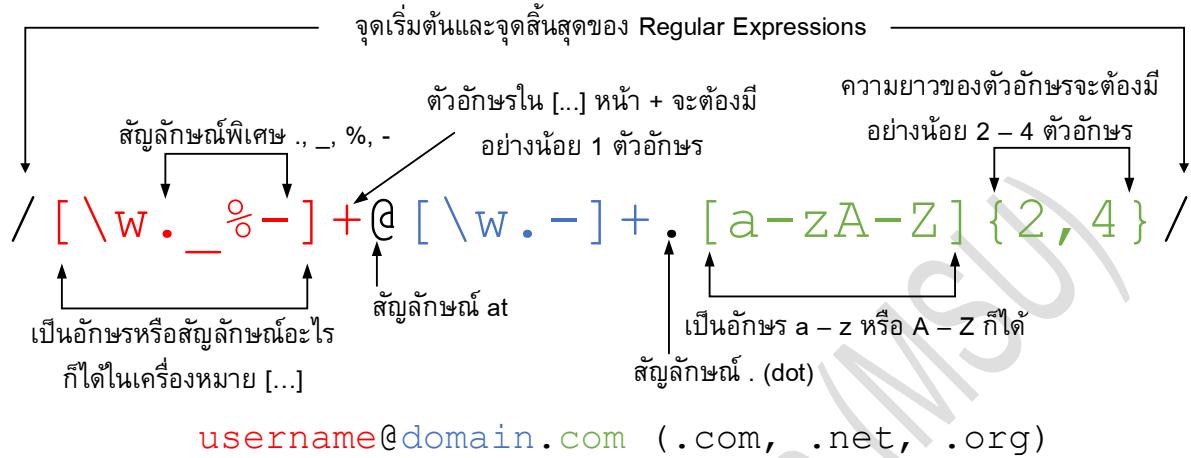
บทที่ 17 โครงสร้างข้อมูลเบื้องต้น

บทที่ 18 ป กิ ณ กะ โ ร แก ร մ มิ ง

บทที่ 12

นิพจน์ปกติ และการประมวลผลข้อความ

(Regular Expressions and Text Processing)



1. นิพจน์ปกติคืออะไร?

นิพจน์ปกติ (Regular Expression: Regex) คือ การตรวจสอบรูปแบบการเรียงตัวของตัวอักษร (Pattern) ในข้อความหรือสตริงว่ามีลักษณะเป็นอย่างไร ซึ่งการใช้ Regex จะทำให้ผู้เขียนโปรแกรมสามารถกำหนดรูปแบบในการค้นหา รวมทั้งสามารถจัดการกับข้อมูล (โดยเฉพาะอย่างยิ่งข้อมูลที่เป็น สตริงและ Text) ได้อย่างมีประสิทธิภาพมากขึ้น

ตัวอย่างการค้นหารูปแบบการเรียงตัวของตัวอักษรที่พบบ่อยคือ ที่อยู่อาศัย (Address) ที่มีการ เรียงตัวของตัวอักษรเป็นข้อความติดต่อกันยาวๆ เช่น

"19/988 หมู่ 5 ถนนเพชรบุรีตัดใหม่ แขวงบางไผ่ เขตบางแค กรุงเทพฯ 10160"

"73/184 หมู่ 5 ถนนเพชรบุรีตัดใหม่ แขวงบางกะปิ เขตหัวหมาก กรุงเทพฯ 10320"

จากรูปแบบข้อความที่อยู่อาศัยด้านบน สามารถจำแนกรูปแบบการเรียงตัวของตัวอักษรคร่าวๆ ได้ดังนี้

1. เลขที่บ้านจะอยู่ข้างหน้าเสมอและจะต้องมีตัวเลขอย่างน้อยหนึ่งตัว ตามด้วยเครื่องหมาย "/" หลังเครื่องหมายดังกล่าวต้องมีตัวเลขอย่างน้อย 1 ตัว เช่น 19/988
2. "หมู่" หมายถึง หมู่ที่ เช่น หมู่ 5
3. "ถนน" หมายถึง ถนน เช่น ถนนเพชรบุรีตัดใหม่
4. "แขวง" และ "เขต" ความหมายถึงชื่อของแขวงหรือเขตที่อาศัยอยู่ เช่น แขวงบางไผ่
5. รหัสไปรษณีย์ เป็นตัวเลขห้าหลักอยู่หลังสุด เช่น 10160

เมื่อผู้เขียนโปรแกรมต้องการตรวจสอบการเรียงตัวของตัวอักษรว่า "มีที่อยู่อาศัยใดบ้างที่มีบ้านเลขที่ ตั้งแต่ 1 - 20, หมู่ 1 – 10, จังหวัดกรุงเทพฯ บ้าง" สามารถเขียนเป็น Regex ได้ดังนี้

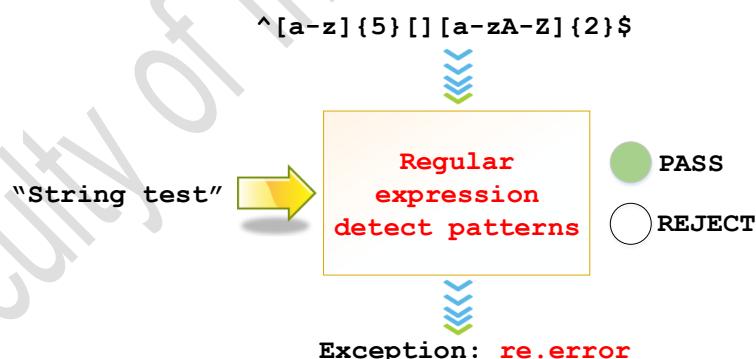
```
regex = r"[0-9]+/[0-9]+ ม.* กรุงเทพฯ [0-9]{5}"
```

โดย $[0 - 9]^+$ หมายถึง เลข 0 ถึง 9 มีจำนวนมากกว่า 1 ตัวขึ้นไป, $*$ หมายถึงตัวเลขหรือตัวอักษรใดๆ ก็ได้และมีจำนวนกี่ตัวก็ได้, $[0 - 9][5]$ รหัสไปรษณีย์เป็นตัวเลข 0 ถึง 9 และต้องมีจำนวน 5 ตัวเท่านั้น เขียนเป็น regex ในภาษา Python ดังนี้

```
regex = r"[1-20]/[0-9]+ ม.[1-10] ถ.* แขวง* เขต* กรุงเทพฯ [0-9]{5}"
```

สำหรับตัวอย่างการใช้งาน Regex กับข้อมูลสารสนเทศอื่นๆ เช่น การค้นหาและแก้ไขคำหรือกลุ่มคำ ด้วยฟังชัน replace ในโปรแกรม Microsoft Word, การใช้ Regex เพื่อตรวจสอบจดหมายอีเล็กทรอนิกซ์ (email) ว่าผู้ใช้งานกรอกข้อมูลชื่อที่อยู่ผู้รับและผู้ส่งถูกต้องตามรูปแบบหรือไม่ และที่นิยมใช้กันมากบน เว็บก็คือ การตรวจสอบข้อมูลบนฟอร์ม เช่น ชื่อผู้ใช้ (Username) ต้องประกอบไปด้วยตัวอักษรตัวเล็ก หรือใหญ่ผสมกับอักษรพิเศษอย่างน้อย 1 ตัว และมีความยาวอยู่ระหว่าง 1 – 15 ตัวอักษร เป็นต้น

ในไฟล่อนร่องรับการใช้งาน Regex ได้อย่างสมบูรณ์แบบ เพราะได้รับการถ่ายทอด ความสามารถดังกล่าวมากจากภาษา Perl นั้นเอง โดยใช้โมดูลชื่อ re จัดการเกี่ยวกับการประมวลผล นิพจน์ปกติ และเมื่อเกิดความผิดพลาดใดๆ เกิดขึ้นจากการประมวลผลนิพจน์ปกติ โปรแกรมจะโยน ความผิดพลาดชื่อว่า re.error ออกมานั่นเอง ดังรูปที่ 12.1



รูปที่ 12.1 แสดงการทำงานของ Regular Expression

2. นิพจน์ปกติอย่างง่าย (Easily Regular Expressions)

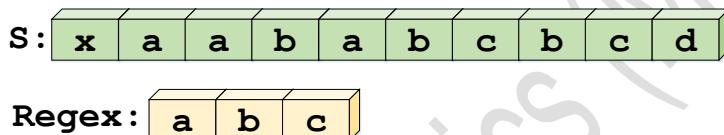
ก่อนอธิบายการใช้งาน Regex อย่างละเอียด ผู้เขียนจะขออธิบายการค้นหาข้อความในข้อความ หรือสตริงก่อน เพราะมีแนวความคิดเหมือนกับ Regex และเพื่อเป็นพื้นฐานให้กับผู้ที่จะศึกษาในเรื่อง Regex ต่อไป

เมื่อผู้เขียนโปรแกรมต้องการค้นหาหรือตรวจสอบคำใดๆ ว่าอยู่ในข้อความหรือไม่ ไฟชอนจะอาศัยคำสั่ง `in` ในการตรวจสอบ ตัวอย่างเช่น ถ้าตัวแปร `s` มีค่าเท่ากับ "This is the Python language." และต้องการค้นหาคำว่า Python ว่าอยู่ในตัวแปรดังกล่าวหรือไม่ ในไฟชอนจะใช้คำสั่งคือ

```
>>> s = "This is the Python language."
>>> "Python" in s
True
```

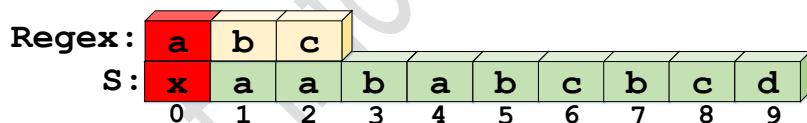
ผลลัพธ์ที่ได้จากตัวอย่างข้างบนคือ `True` เพื่อให้ผู้อ่านเข้าใจการทำงานของการตรวจสอบข้อความเพิ่มขึ้น ผู้เขียนจะขอยกตัวอย่างโดยอาศัยภาพประกอบดังต่อไปนี้

สมมุติว่าสตริงที่ถูกตรวจสอบคือ `S = "xaababcbcd"` และคำที่ใช้สำหรับตรวจสอบคือ `Regex = "abc"` (หรือมองว่าเป็น Regex อย่างง่ายก็ได้) แสดงดังรูปที่ 12.2



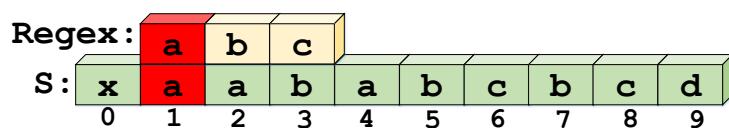
รูปที่ 12.2 แสดงข้อมูลที่เก็บในสตริง `S` และคำที่ใช้ในการตรวจสอบคือ "abc"

ในขั้นตอนแรกของการตรวจสอบ ไฟชอนจะนำข้อมูลในตำแหน่งแรกของสตริงและคำมาเปรียบเทียบว่า เหมือนกันหรือไม่ (แมตช์: Match) ดังนี้ `S[0] == Regex[0]`? ดังแสดงรูปที่ 12.3



รูปที่ 12.3 แสดงการเปรียบเทียบข้อมูลในตำแหน่งที่ [0]

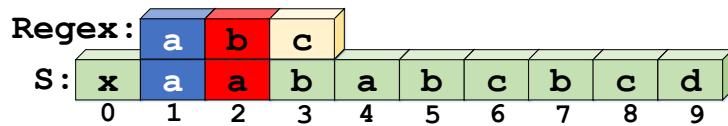
จากรูป 12.3 สีแดงหมายถึง ตำแหน่ง (0) ที่ทำการเปรียบเทียบระหว่างสตริง `S[0]` กับ `Regex[0]` ผลจากการเปรียบเทียบคือ มีค่าไม่เหมือนกัน (Mismatch) โปรแกรมจะเลื่อนตำแหน่งการเปรียบเทียบไปยังตำแหน่งถัดไปคือ `S[1]` กับ `Regex[1]` ดังรูปที่ 12.4



รูปที่ 12.4 แสดงการเปรียบเทียบข้อมูลในตำแหน่งที่ [1]

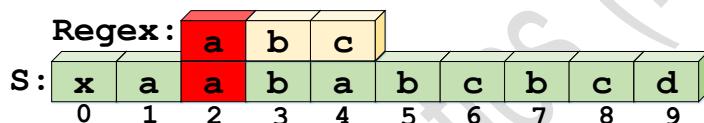
จากรูปที่ 12.4 หลังจากที่โปรแกรมเปรียบเทียบข้อมูลในตำแหน่งที่ 0 และพบว่าข้อมูลทั้งในตัวแปร `S` และ `Regex` ไม่เหมือนกัน ลำดับต่อไปโปรแกรมจะเลื่อนไปทำการเปรียบเทียบข้อมูลในตำแหน่งที่ 1 - 3 (`S[1:4]`) ของตัวแปร `S` ต่อไป ผลลัพธ์จากการเปรียบเทียบในตำแหน่งที่ 1 นี้ ให้ผลลัพธ์

เหมือนกันคือ $S[1] = "a"$ และ $\text{Regex}[1] = "a"$ ผู้เขียนจะใช้สีฟ้า (ตำแหน่งที่ 1) แสดงถึงข้อมูลทั้งสองเหมือนกัน ดังรูปที่ 12.5



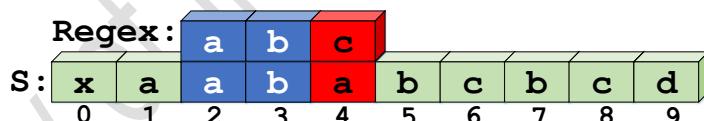
รูปที่ 12.5 แสดงข้อมูลที่ตรงกัน (match) ในตำแหน่งที่ [2]

จากรูปที่ 12.5 เมื่อโปรแกรมเปรียบข้อมูลในตำแหน่งที่ 1 แล้วผลปรากฏว่าข้อมูลเหมือนกัน โปรแกรมจะทำการเก็บสถานะ (สีน้ำเงิน) ไว้ ลำดับถัดไปโปรแกรมจะเลื่อนไปเปรียบเทียบข้อมูลในตำแหน่งที่ 2 ต่อไป แต่ผลปรากฏว่าในตำแหน่งที่ 2 ของ $S[2]$ และ $\text{Regex}[2]$ ไม่เหมือนกัน โปรแกรมจะทำการเลื่อนข้อมูลทั้งชุดของตัวแปร Regex ไปในตำแหน่งที่ 2 ดังรูปที่ 12.6 และเคลียร์สถานะที่เหมือนกัน (สีน้ำเงิน) ทิ้งไปด้วย



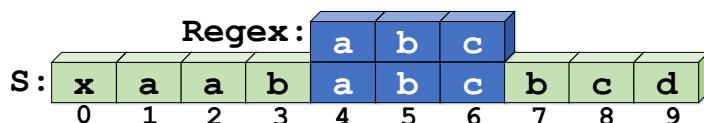
รูปที่ 12.6 แสดงการเลื่อนตำแหน่งของตัวแปร Regex

จากรูปที่ 12.6 โปรแกรมเลื่อนข้อมูลในตัวแปร Regex มาเปรียบเทียบกับข้อมูลในตัวแปร S ตำแหน่งที่ 2 - 4 ($S[2:5]$) ผลจากการเปรียบเทียบ ข้อมูลในตำแหน่งที่ 2 และ 3 มีค่าตรงกัน แต่ข้อมูลในตำแหน่งที่ 4 มีค่าไม่ตรงกัน ดังรูปที่ 12.7



รูปที่ 12.7 แสดงข้อมูลในตำแหน่งที่ 2 และ 3 เหมือนกัน แต่ตำแหน่งที่ 4 ไม่เหมือนกัน

จากรูปที่ 12.7 เมื่อข้อมูลในตำแหน่งที่ 4 ไม่ตรงกัน โปรแกรมจะทำการเลื่อนข้อมูลในตัวแปร Regex ไปอีก 1 ตำแหน่ง ซึ่งจะเริ่มเปรียบเทียบกับข้อมูล S ในตำแหน่งที่ 3 - 5 ($S[3:6]$) ซึ่งผลลัพธ์ที่ได้ก็ไม่เหมือนกันทั้ง 3 ตำแหน่งเช่นเดิม โปรแกรมจะเลื่อนการเปรียบไปเรื่อยๆ จนถึงตำแหน่งที่ $S[4]$ ซึ่งค่าข้อมูลในตัวแปรทั้งสองเหมือนกันทุกประการ แสดงว่าข้อมูลในตัวแปร Regex เป็นสมาชิกหรืออยู่ในตัวแปร S นั่นเอง ดังรูปที่ 12.8



รูปที่ 12.8 แสดงข้อมูลในตัวแปร $S[4:7] == \text{Regex}[0:3]$

3. นิพจน์ปักติในไพธอน (Regular Expressions in Python)

จากตัวอย่างข้างบนแสดงตัวอย่างนิพจน์ปักติอย่างง่าย ในหัวข้อนี้จะกล่าวถึงการใช้งานนิพจน์ปักติในภาษาไพธอน ซึ่งตัวไพธอนเองได้เตรียมโมดูลสำหรับจัดการเกี่ยวกับนิพจน์ปักติ (Regex) ไว้ให้แล้วคือ โมดูล re ซึ่งก่อนการใช้งาน Regex ผู้เขียนโปรแกรมจะต้อง import re เข้ามา ก่อนเสมอ

สำหรับการใช้งาน Regex ในภาษาอื่นๆ ผู้เขียนโปรแกรมจำเป็นต้องเขียน Regex ภายในเครื่องหมาย "/"..."/" ก่อน เช่น ภาษา Perl, SED หรือ AWK เป็นต้น แต่ไพธอนไม่จำเป็นต้องใช้เครื่องหมายดังกล่าว โดยไพธอนมองว่า Regex เป็นสตริงธรรมดائع แต่ติดปัญหาในการใช้งานกับเครื่องหมาย \ (backslash) เล็กน้อย เนื่องจากเครื่องหมายนี้ไพธอนใช้เป็นสัญลักษณ์พิเศษในการพิมพ์ เช่น \t, \n หรือ \t เป็นต้น และใช้คำเนินการกับ Regex ด้วย เพื่อป้องกันการใช้งานเครื่องหมาย \" ไม่ให้ผิดพลาด ไพธอนจะใช้ \"\" แทน \"\" แต่จะประสบปัญหาตามมาอีกคือ ถ้าต้องการใช้ \"\" หลายๆ อันติดกัน เช่น ถ้าต้องการให้เครื่องหมายดังกล่าวติดกัน 2 อันจะใช้คำสั่ง \"\\\"\"\" ผลลัพธ์ที่ได้คือ \"\"\" ทำให้ยุ่งยากกับการใช้งานมาก ซึ่งผู้เขียนโปรแกรมมีทางเลือกที่ดีกว่าคือ ใช้รหัสคำสั่ง raw string ซึ่งคำสั่งดังกล่าวจะทำให้ไพธอนไม่ตีความหมายของเครื่องหมายพิเศษใดๆ ทั้งสิ้น (มองทุกตัวอักษรเป็นอักษรแบบธรรมดากัน) ดังนี้

```
>>> print(r"C:\\\\programs")
C:\\\\programs
>>> print(r"^.*\.\html$")
^.*\.\html$
```

จากตัวอย่างข้างบน raw string (สตริงดิบ) จะต้องขึ้นต้นด้วยตัวอักษร 'r' ก่อน Regex ที่ต้องการใช้งานเสมอ จากผลลัพธ์ดังกล่าวแสดงให้เห็นว่าเครื่องหมายพิเศษต่างๆ เช่น \, *, ^ หรือ \$ จะถูกมองว่าเป็นตัวอักษรธรรมดานั้น จะทำให้ผู้เขียนโปรแกรมไม่จำเป็นต้องกังวลเกี่ยวกับสัญลักษณ์พิเศษที่ใช้ควบคุมในภาษาไพธอนเลย

4. ไวยกรณ์ของนิพจน์ปักติ (Syntax of Regular Expressions)

นิพจน์ปักติอย่างง่าย เช่น r"cat" สามารถเข้าคู่ หรือแมตช์ (Match) ได้กับสตริงมากมาย ตัวอย่างเช่น "a cat and a rat can't be friends.", "Education", "Communicate", "Falsification", "Ramifications", "Cattle" และอีกมากมายเป็นต้น แต่ถ้าผู้เขียนโปรแกรมต้องการให้ r"cat" แมตช์เฉพาะ กับข้อความ "a cat and a rat can't be friends." จะต้องทำอย่างไร? ปัญหานี้เรียกว่า "Over matching" หรือแมตช์เกินความจำเป็นจากที่ต้องการ สำหรับปัญหานี้สามารถแก้ไขให้ถูกต้องโดยเปลี่ยนจาก r"cat"

เป็น `r" cat "` (มีอักษรว่างด้านซ้ายและขวาของคำ) เมื่อต้องการเขียนไฟชอนให้ตรวจสอบข้อความดังกล่าวสามารถทำได้ดังนี้คือ

```
>>> import re
>>> x = re.search("cat", "A cat and a rat can't be friends.")
>>> print(x)
<_sre.SRE_Match object; span=(2, 5), match='cat'>
>>> x = re.search("cow", "A cat and a rat can't be friends.")
>>> print(x)
None
```



Tips: `span=(2, 5)` หมายถึง ตำแหน่งที่คันப์คำในข้อความ จากตัวอย่างคือ ตำแหน่งตั้งแต่ 2 ถึง 4 (`A` = ตำแหน่ง 0, `'` = ตำแหน่ง 1, `c` = 2, `a` = 3 และ `t` = 4)

จากตัวอย่างโปรแกรมข้างต้น โปรแกรมเริ่มต้นด้วยการ `import re` เข้ามำทำงานกับ Regex โดยเรียกเมธอดชื่อ `search()` ซึ่งเป็นเมธอดที่มีความสำคัญและใช้งานบ่อยที่สุด มีรูปแบบคือ `re.search(regex, s)` โดย `regex` คือ นิพจน์ปกติ และ `s` คือสตริงที่ต้องการเปรียบเทียบ คำสั่งแรกเป็นการเปรียบเทียบว่า `"cat"` อยู่ในข้อความ `"a cat and a rat can't be friends."` หรือไม่ ผลลัพธ์ที่ได้จะเป็น `True` หรือ `False` แต่จะเป็นอ็อปเจกต์แทน (ไม่เหมือนกับภาษาอื่นๆ ที่ส่วนใหญ่จะส่งค่ากลับเป็นจริง หรือเท็จ) เมื่อทำการเปรียบเทียบใหม่อีกรอบโดยใช้คำว่า `"cow"` กับข้อความเดิม ผลลัพธ์ที่ได้รับคือ `None` ผู้เขียนโปรแกรมจำเป็นต้องใช้คำสั่ง `if` เพื่อช่วยในแสดงผลลัพธ์ดังนี้

```
import re
if re.search("cat", "A cat and a rat can't be friends."):
    print("Some kind of cat has been found!!")
else:
    print("Some kind of cat has been found!!")

if re.search("cow", "A cat and a rat can't be friends."):
    print("Cats and Rats and a cow!!")
else:
    print("No cow around!!")
```



Some kind of cat has been found!!
No cow around!!

1. สัญลักษณ์ที่ใช้แทนตัวอักษรใดๆ เพียง 1 ตัว(.)

เมื่อผู้เขียนโปรแกรมต้องการอ้างถึงตัวอักษรใดๆ 1 ตัว ในนิพจน์ปกติ จะใช้อักษร '.' แทน เช่น `".at"` หมายถึง ขึ้นต้นด้วยตัวอักษรอะไรก็ได้ 1 ตัว และต้องลงท้ายด้วย `"at"` ตัวอย่างของข้อความที่เป็นไปได้คือ `"rat"`, `"cat"`, `"bat"`, `"eat"`, `"sat"`, `"1at"`, `"&at"` เป็นต้น เขียนเป็น raw string ได้ดังนี้

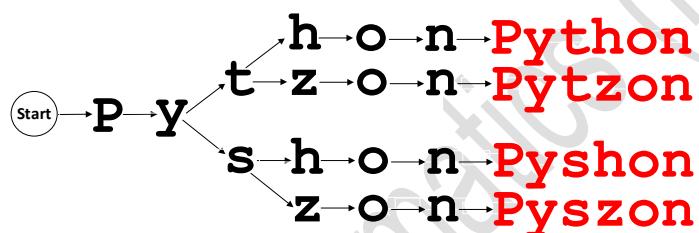
```
s = r".at"
```



Note: อักขระหมายถึง สัญลักษณ์ใดๆ ที่ใช้สำหรับสื่อสารหรือแทนความหมายอย่างโดยย่างหนึ่งตามที่ได้ตกลงกันไว้ (สัญลักษณ์ใดๆ ทั้งหมดที่ใช้ในโลกถือเป็นอักขระทั้งหมด), ตัวอักษรหมายถึง ตัวอักษร a – z, A – Z และ '_' (underscore), ตัวเลขหมายถึง 0 ถึง 9

2. สัญลักษณ์ที่ใช้เลือกตัวอักษรได้ 1 ตัว จากกลุ่มของอักษร ([...])

เครื่องหมาย [...] หมายถึง สามารถเลือกตัวอักษรตัวใดตัวหนึ่งได้เพียง 1 ตัวเท่านั้น จากตัวอักษรทั้งหมดที่อยู่ภายในเครื่องหมายดังกล่าว เช่น ถ้ากลุ่มของตัวอักษรประกอบด้วย [xy1@] ดังนั้นสามารถเลือกได้เฉพาะ "x" หรือ "y" หรือ "@" ตัวใดตัวหนึ่งได้เพียงตัวเดียวเท่านั้น เช่น raw string ของตัวแปร s = r"Py[ts][hz]on" ตัวแปรดังกล่าวสามารถเขียนเป็นข้อความทั้งหมดได้ 4 แบบคือ "Python", "Pytzon", "Pyshon" และ "Pyszon" เมื่อเขียนเป็นผังจำลองการทำงานได้ดังรูปที่ 12.9



รูปที่ 12.9 แสดงผังจำลองการทำงานของ r"Py[ts][hz]on"

จากรูปที่ 12.9 แสดงตัวอย่างการเลือกตัวอักษรเพียง 1 ตัวจากกลุ่มของตัวอักษรที่อยู่ภายในเครื่องหมาย [...] เมื่อผู้เขียนโปรแกรมต้องการขยายขอบเขตของข้อมูลให้ครอบคลุมตัวอักษรมากขึ้น สามารถใช้สัญลักษณ์ '-' แทนได้ แต่ข้อมูลดังกล่าวต้องเป็นมีค่าข้อมูลที่ต่อเนื่องกัน เช่น 0 ถึง 9, a ถึง z, A ถึง Z เช่น [a-e] ข้อมูลที่ถูกเลือกคือ a, b, c, d, หรือ e ตัวใดตัวหนึ่ง หรือ [0-5] ข้อมูลที่ถูกเลือกคือ 0, 1, 2, 3, 4 หรือ 5 เป็นต้น และเมื่อต้องการกำหนดให้ Regex เป็นทั้งอักษรตัวเล็กและตัวใหญ่สามารถเขียนเป็น Regex ได้ดังนี้คือ [a-zA-Z] หรือสามารถสมตัวอักษรรับตัวเลขได้เช่น [a-zA-Z0-9] ซึ่งหมายถึง ข้อมูลตัวใดตัวหนึ่งที่อยู่ระหว่าง a ถึง z หรือ A ถึง Z หรือ 0 ถึง 9 เป็นต้น



Note: จำไว้เสมอว่าตัวอักษรที่อยู่ภายในเครื่องหมาย [...] จะถูกเลือกไปใช้งานเพียงตัวเดียวเท่านั้น

สัญลักษณ์ '-' ยังมีความพิเศษที่แตกต่างออกไป เมื่อใช้เครื่องหมายดังกล่าววางแผนอยู่ด้านหน้าและด้านหลังตัวอักษรได้ (อยู่ตัดจากเครื่องหมาย [หรืออยู่ก่อนหน้าเครื่องหมาย]) ใน [...] เช่น [-az] หมายถึง ข้อมูลจะถูกเลือกระหว่าง '-', 'a' หรือ 'z' และถ้ากำหนดเป็น [az-] ก็จะได้ผลลัพธ์เช่นเดียวกัน สำหรับตัวอย่างการใช้ [...] และ '-' ดังนี้

[th] = 't' หรือ 'h' ตัวใดตัวหนึ่งเท่านั้น
[0123] = '0', '1', '2' หรือ '3' ตัวใดตัวหนึ่งเท่านั้น
[ab12] = 'a', 'b', '1' หรือ '2' ตัวใดตัวหนึ่ง
[a-c0-2] = 'a', 'b', 'c', '0', '1' หรือ '2' ตัวใดตัวหนึ่ง
[a-cA-C0-2] = 'a', 'b', 'c', 'A', 'B', 'C', '0', '1' หรือ '2' ตัวใดตัวหนึ่ง
[-3] = '-' หรือ '3' ตัวใดตัวหนึ่ง
[-a-c] = '!', 'a', 'b' หรือ 'c' ตัวใดตัวหนึ่ง
[a-c-] = '!', 'a', 'b' หรือ 'c' ตัวใดตัวหนึ่ง
[g-r] = 'g' ถึง 'r' ตัวใดตัวหนึ่ง

สัญลักษณ์ '^' (Caret) เมื่อใส่ไว้หลังเครื่องหมาย '[' และนำหน้าตัวอักษรได เช่น [^abc] แสดงว่า ข้อมูลหลังสัญลักษณ์ดังกล่าวจะถูกยกเว้น หรือไม่ถูกเลือก ดังตัวอย่าง

[^abc] = ตัวอักษร 'a', 'b' หรือ 'c' จะไม่ถูกเลือก
[^13579] = ตัวเลข '1', '3', '5', '7' หรือ '9' เท่านั้นที่จะไม่ถูกเลือก
[^ศສษ] = ตัวอักษรภาษาไทย 'ศ', 'ສ' หรือ 'ษ' เท่านั้นที่จะไม่ถูกเลือก

เมื่อสัญลักษณ์ '^' อยู่หลังจากตัวอักษรจะมีความหมายเป็นตัวอักษรธรรมดานะเท่านั้น เช่น [a^bc] หมายถึง 'a', '^', 'b' หรือ 'c' ตัวใดตัวหนึ่งจะถูกเลือก หรือ [abc^], [ab^c] จะมีความหมายเช่นเดียวกับ [a^bc] ตัวอย่างโปรแกรมที่ 12.1 แสดงการใช้งาน Regex ในการค้นหาหมายเลขโทรศัพท์ที่เก็บในแฟ้มชื่อ phone_book.txt



Input File: phone_book.txt

```
Allison Neu 555-8396
Bob Newhall 555-4344
C. Montgomery Burns 555-0001
C. Montgomery Burns 555-0113
Canine College 555-7201
Canine Therapy Institute 555-2849
Cathy Neu 555-2362
...
The Nuclear Powerplant 555-5246
The Simpsons' residence 555-8707
The Simpsons, 742 Evergreen Terrace 555-0113
Toby Muntz 555-9972
```

Program Example 12.1: searching phone number by using regex

```
⇒1 import re
⇒2 f = open("phone_book.txt")
⇒3 for line in f:
⇒4     if re.search(r"J.*Neu", line):
```

```

⇒5     print(line.rstrip())
⇒6     f.close()

```



OUTPUT

```

Jack Neu 555-7666
Jeb Neu 555-5543
Jennifer Neu 555-3652

```

จากตัวอย่างโปรแกรมที่ 12.1 แสดงการสร้าง Regex เพื่อค้นหาหมายเลขโทรศัพท์ที่ขึ้นต้นด้วยตัวอักษร J ตัวใหญ่ ⇒ ตามด้วยอักษรใดๆ ก็ได้แบบไม่จำกัดความยาว (ใช้สัญลักษณ์ .*) ⇒ และลงท้ายด้วย "Neu"

เริ่มต้นบรรทัดที่ 1 โปรแกรมนำเข้าโมดูลชื่อ re เพื่อใช้สำหรับประมวลผล Regex ในบรรทัดที่ 2 โปรแกรมทำการเปิดแฟ้มชื่อ phone_book.txt ซึ่งเก็บรายชื่อและหมายเลขโทรศัพท์ของผู้ใช้เอาไว้ อ้อมเจก์ของแฟ้มที่สร้างขึ้นจะเก็บไว้ในตัวแปรชื่อ f (ถ้าผู้อ่านยังไม่มีพื้นฐานด้านการเปิดปิดแฟ้มให้กลับไปอ่านในบทที่ 10 ก่อน) ขั้นตอนต่อไปบรรทัดที่ 3 โปรแกรมทำการอ่านข้อมูลในสมุดโทรศัพท์มาครั้งละ 1 บรรทัดด้วยคำสั่ง for ข้อมูลที่อ่านได้ในแต่ละบรรทัดจะถูกส่งให้กับเมธอด search() เพื่อค้นหาชื่อผู้ใช้โทรศัพท์ที่ต้องการ (บรรทัด 4) โดยเมธอดดังกล่าวต้องการพารามิเตอร์ 2 ตัวคือ Regex ที่มีค่าเท่ากับ r"J.*Neu" และข้อมูลของแต่ละบรรทัดในสมุดโทรศัพท์ ผลลัพธ์จากเมธอด search() จะถูกส่งให้กับคำสั่ง if ตรวจสอบว่าคันพบหรือไม่ (ถ้าพบชื่อผู้ใช้ที่ต้องการค่าที่ส่งกลับจากเมธอด search() จะมีค่าไม่เท่ากับ None) เมื่อผลลัพธ์ที่เปรียบเทียบด้วยคำสั่ง if เป็นจริง โปรแกรมจะสั่งพิมพ์ข้อความดังกล่าวออกจากภาพ (บรรทัด 5) แต่ก่อนการพิมพ์โปรแกรมจะทำการตัดบรรทัดว่างออกจากข้อความที่จะพิมพ์ก่อนด้วยคำสั่ง line.rstrip() ในบรรทัดสุดท้าย (บรรทัด 6) โปรแกรมทำการสั่งปิดแฟ้มข้อมูลและจบโปรแกรม

3. สัญลักษณ์พิเศษอื่นๆ ที่ใช้สร้าง Regex

นอกเหนือจากเครื่องหมาย '-' และ '.' ที่ทำงานร่วมกับสัญลักษณ์ [...] และ '+' พร้อมยังได้จัดเตรียมสัญลักษณ์พิเศษอื่นๆ เพื่อช่วยอำนวยความสะดวกในการสร้าง Regex ดังนี้

กำหนดให้ regex = r"[a-zA-Z0-9_]" เมื่อดำเนินการกับสัญลักษณ์พิเศษต่างๆ ต่อไปนี้แล้ว ผลลัพธ์ที่ได้คือ

- \d แมตช์กับตัวเลขฐานสิบตัวใดตัวหนึ่ง คือชุดข้อมูล [0-9] หรือสามารถเขียน regex ใหม่คือ r"[a-zA-Z\d_]" เช่น

```

>>> import re
>>> regex = r"[a-zA-Z\d]"
>>> print(re.search(regex, "9"))
<sre.SRE_Match object; span=(0, 1), match='9'>
>>> print(re.search(s, "*"))

```

None

```
>>> print(re.search(regex, "a"))
<sre.SRE_Match object; span=(0, 1), match='a'>
```



Note: <sre.SRE_Match object...> หมายถึงแมตช์ และ **None** หมายถึง ไม่แมตช์

- \D ทำงานตรงกันข้ามกับ \d คือแมตช์กับอักษรใดๆ ก็ได้ที่ไม่ใช่เลขฐานสิบ เมื่อเปรียบเทียบกับ regex จะเทียบเท่ากับชุดข้อมูล [^0-9] หรือสามารถเขียน regex ใหม่คือ r"[a-zA-Z\D]" หรือ r" a-zA-Z\D"

```
>>> import re
>>> regex = r"[a-zA-Z\D]"
>>> print(re.search(regex, "9"))
None
>>> print(re.search(regex, "*"))
<sre.SRE_Match object; span=(0, 1), match='*'>
>>> print(re.search(regex, "a"))
<sre.SRE_Match object; span=(0, 1), match='a'>
```

- \s แมตช์กับตัวอักขระว่างตัวใดตัวหนึ่ง เทียบเท่ากับชุดข้อมูล คือ [\t\n\r\f\v] หรือ r"\s" หรือ r"\s" เช่น

```
>>> import re
>>> regex = r"\s"
>>> re.search(regex, " ")
<sre.SRE_Match object; span=(0, 1), match=' '>
>>> re.search(regex, "\t")
<sre.SRE_Match object; span=(0, 1), match='\t'>
>>> re.search(regex, "\n")
<sre.SRE_Match object; span=(0, 1), match='\n'>
>>> re.search(regex, "\r")
<sre.SRE_Match object; span=(0, 1), match='\r'>
>>> re.search(regex, "\f")
<sre.SRE_Match object; span=(0, 1), match='\f'>
>>> re.search(regex, "\v")
<sre.SRE_Match object; span=(0, 1), match='\v'>
```

- \S ทำงานตรงกันข้ามกับ \s คือแมตช์ตัวอักขระที่ไม่ใช่ค่าว่างตัวใดตัวหนึ่ง เทียบเท่ากับชุดข้อมูลคือ [^\t\n\r\f\v] หรือ r"\S" หรือ r"\S" เช่น

```
>>> import re
>>> regex = r"\S"
>>> re.search(regex, "a")
<sre.SRE_Match object; span=(0, 1), match='a'>
>>> re.search(regex, "\\")
<sre.SRE_Match object; span=(0, 1), match='\\'>
>>> re.search(regex, "&")
<sre.SRE_Match object; span=(0, 1), match='&'>
```

```
>>> print(re.search(regex, "\n"))
None
```

- **\w** แมตช์กับตัวอักษร หรือตัวเลข หรือสัญลักษณ์ '_' (Underscore) ตัวใดตัวหนึ่ง คือเท่ากับชุดข้อมูล [a-zA-Z0-9_] หรือสามารถเขียน regex ใหม่คือ r"\w" หรือ r"\W"

```
>>> import re
>>> regex = r"\w"
>>> print(re.search(regex, "a"))
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>> print(re.search(regex, "A"))
<_sre.SRE_Match object; span=(0, 1), match='A'>
>>> print(re.search(regex, "_"))
<_sre.SRE_Match object; span=(0, 1), match='_ '>
>>> print(re.search(regex, "&"))
None
```

- **\W** ทำงานตรงกันข้ามกับ \w คือแมตช์กับสัญลักษณ์ที่ไม่ใช่ทั้งตัวอักษร หรือตัวเลข หรือ '_' ตัวใดตัวหนึ่ง เช่น " " (ข้อความว่าง), "@", "\n", "*", "+" เป็นต้น

```
>>> import re
>>> regex = r"\W"
>>> print(re.search(regex, "a"))
None
>>> print(re.search(regex, "_"))
None
>>> print(re.search(regex, " "))
<_sre.SRE_Match object; span=(0, 1), match=' ' >
>>> print(re.search(regex, "@"))
<_sre.SRE_Match object; span=(0, 1), match='@'>
>>> print(re.search(regex, "\n"))
<_sre.SRE_Match object; span=(0, 1), match='\n'>
```

- **\b** แมตช์กับคำระหว่าง \W กับ \w หรือระหว่าง \w กับ \W เช่น r'\bfoo\b' แมตช์กับข้อความ 'foo', 'foo.', '(foo)' หรือ 'bar foo baz' แต่ไม่แมตช์กับ 'foobar' หรือ 'foo3'



Tips: \b จะแมตช์ก็ต่อเมื่อตัวอักษรปกติ (0-9, a-z, A-Z, underscore '_') อยู่ติดกับอักษรพิเศษ (' ', @, *, \n, !) แต่จะไม่แมตช์กับตัวอักษรปกติกับปกติ หรืออักษรพิเศษกับพิเศษ

- **\B** แมตช์กับคำระหว่าง \W กับ \W หรือระหว่าง \w กับ \w เช่น r'py\B' จะแมตช์กับข้อความ 'python', 'py3', 'py_' แต่จะไม่แมตช์กับ 'py', 'py.', หรือ 'py!'

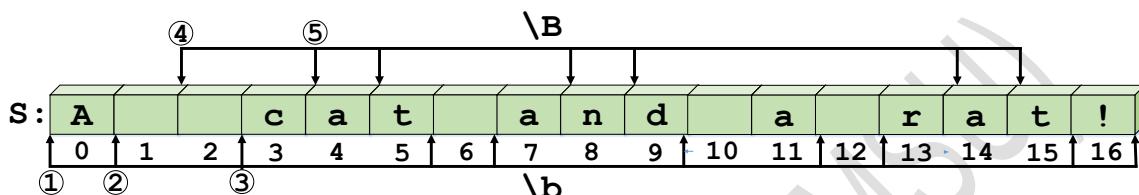


Tips: \B จะแมตช์ก็ต่อเมื่อตัวอักษรปกติ (0-9, a-z, A-Z, underscore '_') อยู่ติดกับอักษรปกติ แต่จะไม่แมตช์กับตัวอักษรปกติกับอักษรพิเศษ

สมมุติให้ `s = "A cat and a rat!"`

ตัวอักษรที่แมตช์กับ `\w` คือ 'A', 'c', 'a', 't', 'a', 'n', 'd', 'a', 'r', 'a', 't' ตามลำดับ

ตัวอักษรที่ไม่แมตช์กับ `\W` คือ ' ', ' ', ' ', ' ', ' ' และ '!' ซึ่งแสดงตัวอย่างการทำงานของ `\b` และ `\B` ในรูปที่ 12.10



รูปที่ 12.10 แสดงการทำงานของ `\b` และ `\B`

สำหรับ `\b` จะแมตช์กับอักษรระหว่าง `\w` กับ `\W` หรือ `\W` กับ `\w` ดังนี้

จากรูปที่ 12.10 อักษรแรกในข้อความนี้คือ 'A' สัญลักษณ์ `\b` จะแมตช์กับอักษรระหว่าง `\w` และ `\W` ดังนั้นด้านซ้ายของ 'A' ❶ ถือว่าเป็น `\W` และตัวอักษร 'A' ถือว่าเป็น `\w` ลูกศรจี้ชี้ที่ด้านซ้ายของ 'A'

ลำดับถัดมา ❷ ทำการเปรียบเทียบตัวอักษร 'A' (`\w`) กับสตริงว่าง (`\W`) ด้านซ้ายของ 'A' ซึ่งตรงตามเงื่อนไขคือ อยู่ระหว่าง `\w` กับ `\W` จะนั่นลูกศรจี้ชี้ที่ด้านขวาของ 'A'

ลำดับถัดมา ❸ ทำการเปรียบเทียบช่วงว่างด้านซ้ายของ 'c' (`\W`) กับตัวอักษร 'c' (`\w`) ซึ่งตรงตามเงื่อนไขคือ อยู่ระหว่าง `\w` กับ `\w` จะนั่นลูกศรจี้ชี้ที่ด้านซ้ายของ 'c'

สำหรับ `\B` ตำแหน่งที่เกิดขึ้นระหว่าง `\w` กับ `\w` หรือ `\W` กับ `\W` ดังนี้

❹ ทำการเปรียบเทียบระหว่างอักษรว่าง '' (`\W`) กับอักษรว่าง '' (`\W`) ซึ่งตรงตามเงื่อนไขคือ อยู่ระหว่าง `\w` กับ `\W` จะนั่นลูกศรจี้ชี้ที่ตรงกลางระหว่างอักษรว่าง

❺ ทำการเปรียบเทียบระหว่างตัวอักษร 'c' (`\w`) กับตัวอักษร 'a' (`\w`) ซึ่งตรงตามเงื่อนไขคือ อยู่ระหว่าง `\w` กับ `\w` จะนั่นลูกศรจี้ชี้ที่ตรงกลางระหว่าง 'c' กับ 'a'

- ❻ แมตช์กับเครื่องหมาย '' เช่น

```
>>> s = "C:\Programs\Python"
>>> print(s)
C:\Programs\Python
```

- ❼ (...) ใช้สำหรับกำหนดกลุ่มของคำ เช่น "(a|b)c" หมายถึง มี 2 กลุ่มคำคือ "ac" และ "bc" เช่น

```
>>> import re
>>> regex = r"(a|b)c"
>>> print(re.search(regex, "ac"))
<_sre.SRE_Match object; span=(0, 2), match='ac'>
>>> print(re.search(regex, "bc"))
<_sre.SRE_Match object; span=(0, 2), match='bc'>
```

- ? ตัวอักษรหรือกลุ่มคำที่นำหน้าเครื่องหมายดังกล่าวจะมีหรือไม่มีก็ได้ (Option) เช่น a?b แมตช์กับ b, ab, aaab, aaaab, acb หรือ cccb, cccbbb อย่างใดอย่างหนึ่งก็ได้ เช่น

```
>>> import re
>>> regex = r"a?b"
>>> print(re.search(regex, "a"))
None
>>> print(re.search(regex, "b"))
<_sre.SRE_Match object; span=(0, 1), match='b'>
>>> print(re.search(regex, "aab"))
<_sre.SRE_Match object; span=(1, 3), match='ab'>
>>> print(re.search(regex, "abb"))
<_sre.SRE_Match object; span=(0, 2), match='ab'>
>>> print(re.search(regex, "abbbb"))
<_sre.SRE_Match object; span=(0, 2), match='ab'>
>>> print(re.search(regex, "a$*ab@"))
<_sre.SRE_Match object; span=(3, 5), match='ab'>
```

หรือตัวอย่างเช่น r"Feb(ruary)? 2011" จะให้ผลลัพธ์ดังนี้

```
>>> import re
>>> print(re.search(r"Feb(ruary)? 2011", "Feb"))
None
>>> print(re.search(r"Feb(ruary)? 2011", "2011"))
None
>>> print(re.search(r"Feb(ruary)? 2011", "Feb 2011"))
<_sre.SRE_Match object; span=(0, 8), match='Feb 2011'>
>>> print(re.search(r"Feb(ruary)? 2011", "February 2011"))
<_sre.SRE_Match object; span=(0, 13), match='February
2011'>
```

จากตัวอย่างโปรแกรมด้านบน regex = r"Feb(ruary)? 2011" หมายถึง โปรแกรมจะแมตช์กับข้อความว่า "Feb 2011" หรือ "February 2011" เท่านั้น

- * แมตช์กับตัวอักษรหรือคำใดๆ ที่อยู่หน้าเครื่องหมายดังกล่าว และมีจำนวนกี่ตัวก็ได้ หรือไม่มีก็ได้ เช่น 1*2 แมตช์กับ 12, 112, 122, 11112, 12222, 111111112, 122222222222 เป็นต้น หรือ 1(abc)*2 คือ 12, 1abc2, 1abcabc2, 1abcababcabc2, 1abcababcabcabc2 เป็นต้น

```
>>> import re
>>> print(re.search(r"1*2", "12"))
<_sre.SRE_Match object; span=(0, 2), match='12'>
```

```
>>> print(re.search(r"1*2","112"))
<_sre.SRE_Match object; span=(0, 3), match='112'>
>>> print(re.search(r"1*2","122"))
<_sre.SRE_Match object; span=(0, 2), match='12'>
>>> print(re.search(r"1*2","11112"))
<_sre.SRE_Match object; span=(0, 5), match='11112'>
>>> print(re.search(r"1*2","12222"))
<_sre.SRE_Match object; span=(0, 2), match='12'>
>>> print(re.search(r"1*2","1111111112"))
<_sre.SRE_Match object; span=(0, 10), match='1111111112'>
>>> print(re.search(r"1*2","1222222222"))
<_sre.SRE_Match object; span=(0, 2), match='12'>
>>> print(re.search(r"1*2","1111111111"))
None
>>> print(re.search(r"1*2","2222222222"))
<_sre.SRE_Match object; span=(0, 1), match='2'>
```

หรือ regex = r"1(abc)*2"

```
>>> print(re.search(r"1(abc)*2","12"))
<_sre.SRE_Match object; span=(0, 2), match='12'>
>>> print(re.search(r"1(abc)*2","1abc2"))
<_sre.SRE_Match object; span=(0, 5), match='1abc2'>
>>> print(re.search(r"1(abc)*2","1abcabc2"))
<_sre.SRE_Match object; span=(0, 8), match='1abcabc2'>
>>> print(re.search(r"1(abc)*2","1abcabca2"))
<_sre.SRE_Match object; span=(0, 11),
match='1abcabca2'>
```

- + คำหรือตัวอักษรที่อยู่หน้าเครื่องหมายนี้ จะต้องปรากฏอยู่ในข้อความที่นำมาตรวจสอบอย่างน้อย 1 ตัว เช่น a+ แมตช์กับ abc, a123, 1a23abc, xyz_a, 12aaa45 หรือ x(abc)+y แมตช์กับ xabcabcy, xabcabcabcy หรือ 0(123)+9 แมตช์กับ 01239, 01231239 เป็นต้น

```
>>> import re
>>> print(re.search(r"a+","a"))
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>> print(re.search(r"a+","abc"))
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>> print(re.search(r"a+","a123"))
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>> print(re.search(r"a+","1a23abc"))
<_sre.SRE_Match object; span=(1, 2), match='a'>
>>> print(re.search(r"a+","xyz_a"))
<_sre.SRE_Match object; span=(3, 4), match='a'>
>>> print(re.search(r"x(abc)+y","xabcy"))
<_sre.SRE_Match object; span=(0, 5), match='xabcy'>
>>> print(re.search(r"x(abc)+y","xabcabcy"))
<_sre.SRE_Match object; span=(0, 8), match='xabcabcy'>
>>> print(re.search(r"0(123)+9","01231231239"))
<_sre.SRE_Match object; span=(0, 11),
match='01231231239'>
```

- ^ แมตช์กับคำหรือตัวอักษรที่ขึ้นต้นข้อความที่นำมาตรวจสอบ เช่น ^abc คือ abc123, abcdef, abc\$3s เป็นต้น
- . ใช้แทนตัวอักษรใดๆ เพียง 1 ตัวเท่านั้น ยกเว้น newline (\n) เช่น x.y แมตช์กับ xzy, x1y, xxy หรือ xy แต่จะไม่แมตช์กับ xaby, x\ny หรือ xxxy เป็นต้น

```
>>> import re
>>> regex = r"x.y"
>>> print(re.search(regex, "x"))
None
>>> print(re.search(regex, "y"))
None
>>> print(re.search(regex, "xzy"))
<_sre.SRE_Match object; span=(0, 3), match='xzy'>
>>> print(re.search(regex, "x1y"))
<_sre.SRE_Match object; span=(0, 3), match='x1y'>
>>> print(re.search(regex, "x12y"))
None
>>> print(re.search(regex, "x\ny"))
None
```

ถ้ากำหนดรูปแบบ regex = r"[0-9]." หมายถึง แมตช์กับตัวเลขได้ตัวเลขหนึ่งตัวแต่ 0 ถึง 9 ([0-9]) \Rightarrow และตามด้วยตัวอักษรใดๆ ก็ได้ ต่อท้ายอีกหนึ่งตัว(.) เช่น 00, 01, 11, 25, 58, 99, 012, 0123, 0a, 0ab แต่ไม่แมตช์กับ a1, ab เป็นต้น

ถ้ากำหนดรูปแบบ regex = r".*" หมายถึง แมตช์กับตัวอักษรตัวใดๆ ก็ได้ รวมถึงอักษรพิเศษ และมีจำนวนกี่ตัวก็ได้ (.) เช่น a, ab, abc, abcdef, 1, 12, 123, 123456, aaabbb12*%@%* หรือ พูดง่ายๆ คือ แมตช์กับทุกๆ สิ่ง โดยไม่จำกัดความยาว

ถ้ากำหนดรูปแบบ regex = r"[0-9].*" หมายถึง แมตช์กับตัวเลขได้ตัวเลขหนึ่งตัวแต่ 0 ถึง 9 \Rightarrow และตามด้วยอักษรใดๆ ก็ได้ โดยไม่จำกัดความยาว (.) เช่น 00, 01, 11, 25, 58, 99, 012, 0123, 0a, 0ab, 1@%* แต่ไม่แมตช์กับ a, ab, c12 เป็นต้น

ถ้ากำหนดรูปแบบ regex = r"[0-9][0-9].*" หมายถึง แมตช์กับตัวเลขได้ตัวเลขหนึ่งตัวแต่ 0 ถึง 9 \Rightarrow ตามด้วยตัวเลขได้ตัวเลขหนึ่งตัวแต่ 0 ถึง 9 \Rightarrow ตามด้วยอักษรว่าง 1 ตัวอักษร (" ") \Rightarrow ตามด้วย อักษรอะไรก็ได้และมีความยาวแบบไม่จำกัด (.) เช่น 12, 12 3456, 12 @%*, 12 abc แต่ไม่แมตช์กับ a, ab, 12, a12 เป็นต้น

```
>>> import re
>>> regex = r"[0-9] [0-9] .*"
>>> print(re.search(regex, "1"))
None
>>> print(re.search(regex, "12"))
None
```

```
>>> print(re.search(regex, "12 "))
<sre.SRE_Match object; span=(0, 3), match='12 '>
>>> print(re.search(regex, "12 3456"))
<sre.SRE_Match object; span=(0, 7), match='12 3456'>
>>> print(re.search(regex, "12 @%*"))
<sre.SRE_Match object; span=(0, 6), match='12 @%*'>
>>> print(re.search(regex, "12 abc"))
<sre.SRE_Match object; span=(0, 6), match='12 abc'>
```

ถ้ากำหนดรูปแบบ regex = r"[0-9]+ .*" หมายถึง แมตช์กับตัวเลขใดตัวเลขหนึ่งตั้งแต่ 0 ถึง 9 อย่างน้อยหนึ่งตัว ([0-9]+) \Rightarrow ตามด้วยอักษรว่าง 1 ตัว (" ") \Rightarrow ตามด้วยอักษรอะไรก็ได้และมีความยาวแบบไม่จำกัด (*) เช่น 1 , 1 2, 12 , 1234 56, 1223456 @%*

```
>>> import re
>>> regex = r"[0-9]+ .*"
>>> print(re.search(regex, "1 "))
<sre.SRE_Match object; span=(0, 2), match='1 '>
>>> print(re.search(regex, "1 2"))
<sre.SRE_Match object; span=(0, 3), match='1 2'>
>>> print(re.search(regex, "12 "))
<sre.SRE_Match object; span=(0, 3), match='12 '>
>>> print(re.search(regex, "1234 56"))
<sre.SRE_Match object; span=(0, 7), match='1234 56'>
>>> print(re.search(regex, "1223456 @%*"))
<sre.SRE_Match object; span=(0, 11), match='1223456 @%*'>
```

ถ้ากำหนดรูปแบบ regex = r"^[0-9][0-9][0-9][0-9] [A-Za-z]+" หมายถึง แมตช์กับตัวเลขใดตัวเลขหนึ่งตั้งแต่ 0 ถึง 9 ที่เรียงต่อกัน 4 ตัว โดยต้องอยู่หน้าสุดของข้อความ (^) \Rightarrow ตามด้วยอักษรว่าง 1 ตัว (" ") และ \Rightarrow ตามด้วยตัวอักษร a ถึง z ตัวเล็กหรือใหญ่ก็ได้อย่างน้อย 1 ตัวอักษร ([A-Za-z]+) เช่น 0123 A, 0123 abc, 5678 AbC แต่ไม่แมตช์กับ a5678 abc หรือ 1234Aa เป็นต้น

```
>>> import re
>>> regex = r"^[0-9][0-9][0-9][0-9] [A-Za-z]+"
>>> print(re.search(regex, "1234 A"))
<sre.SRE_Match object; span=(0, 6), match='1234 A'>
>>> print(re.search(regex, "5678 abc"))
<sre.SRE_Match object; span=(0, 8), match='5678 abc'>
>>> print(re.search(regex, "a5678 abc"))
None
```

- {...} แสดงจำนวนตัวอักษรหรือคำที่ต้องการทำซ้ำ เช่น {2} หมายถึง ทำซ้ำอย่างน้อย 2 ครั้ง ดังตัวอย่าง

1{3} คือ ทำซ้ำตัวเลข 1 เท่ากับ 3 ตัวอักษร เช่น 111, 1111, 11112222, 222111333,
@#%111&*+, @#%&*+1111

```

>>> import re
>>> regex = r"1{3}"
>>> print(re.search(regex, "11"))
None
>>> print(re.search(regex, "111"))
<_sre.SRE_Match object; span=(0, 3), match='111'>
>>> print(re.search(regex, "1111"))
<_sre.SRE_Match object; span=(0, 3), match='111'>
>>> print(re.search(regex, "11111122222"))
<_sre.SRE_Match object; span=(0, 3), match='111'>
>>> print(re.search(regex, "2221111333"))
<_sre.SRE_Match object; span=(3, 6), match='111'>
>>> print(re.search(regex, "@#%111&*+"))
<_sre.SRE_Match object; span=(3, 6), match='111'>
>>> print(re.search(regex, "@#%&*+1111"))
<_sre.SRE_Match object; span=(6, 9), match='111'>

```

ab{2} คือ ทำข้าตัวอักษร b (ไม่ใช่ ab) เท่ากับ 2 ตัว เช่น abb, abbb, aaabbb

```

>>> regex = r"ab{2}"
>>> print(re.search(regex, "ab"))
None
>>> print(re.search(regex, "abb"))
<_sre.SRE_Match object; span=(0, 3), match='abb'>
>>> print(re.search(regex, "abbbbb"))
<_sre.SRE_Match object; span=(0, 3), match='abb'>
>>> print(re.search(regex, "aaabbb"))
<_sre.SRE_Match object; span=(2, 5), match='abb'>

```

a(123){2}b คือ ทำข้ากลุ่มตัวเลข 123 เท่ากับ 2 ชุดติดกัน เช่น a123123b

```

>>> regex = r"a(123){2}b"
>>> print(re.search(regex, "a123b"))
None
>>> print(re.search(regex, "a123123b"))
<_sre.SRE_Match object; span=(0, 8), match='a123123b'>
>>> print(re.search(regex, "a123123123b"))
None

```

ab{2,} คือ ทำข้าตัวอักษร b ตั้งแต่ 2 ถึง N ตัว เช่น abb, abbb, abbbbbbb, aaaabbbbbbb, ababababb เป็นต้น

```

>>> regex = r"ab{2,}"
>>> print(re.search(regex, "ab"))
None
>>> print(re.search(regex, "abb"))
<_sre.SRE_Match object; span=(0, 3), match='abb'>
>>> print(re.search(regex, "abbbbb"))
<_sre.SRE_Match object; span=(0, 6), match='abbbbb'>
>>> print(re.search(regex, "aaabbb"))
<_sre.SRE_Match object; span=(2, 6), match='abbb'>

```

```
>>> print(re.search(regex, "abbabbabb"))
<_sre.SRE_Match object; span=(0, 3), match='abb'>
```

$ab\{start, stop\} \Rightarrow ab\{3, 5\}$ คือ ทำซ้ำตัวอักษร b อยู่ระหว่าง 3 ถึง 5 ตัว (ถ้ากลุ่มตัวอักษรมีค่าเท่ากับ bbbbbbbb จะแมตช์กับ bbbbb โดยไม่แมตช์กับ bbb และ bbbb) เช่น abb, abbb, abbbb, abbbbbbb เป็นต้น

```
>>> regex = r"ab\{3,5\}"
>>> print(re.search(regex, "ab"))
None
>>> print(re.search(regex, "abb"))
None
>>> print(re.search(regex, "abbb"))
<_sre.SRE_Match object; span=(0, 4), match='abbb'>
>>> print(re.search(regex, "aaaaabbbbb"))
<_sre.SRE_Match object; span=(4, 9), match='abbbb'>
>>> print(re.search(regex, "aaaaabbbbb"))
<_sre.SRE_Match object; span=(4, 10), match='abbbbb'>
>>> print(re.search(regex, "aaaaabbbbbbbbbbb"))
<_sre.SRE_Match object; span=(4, 10), match='abbbbb'>
>>> print(re.search(regex, "abbabbbabbbbabbbbabbbbabbb"))
<_sre.SRE_Match object; span=(3, 7), match='abbb'>
```

$regex = r"^{[0-9]\{4\}} [A-Za-z]*"$ หมายถึง ขึ้นต้น (^) ด้วยตัวเลข 0 ถึง 9 จำนวน 4 ตัว ($[0-9]\{4\}$) ตามด้วย \Rightarrow อักษรว่าง 1 ตัว (" ") \Rightarrow ตามด้วยตัวอักษรเล็กหรือใหญ่ก็ได้แบบไม่จำกัดจำนวน ($[A-Za-z]^*$) เช่น 1234 Aa, 1234 AZaz, 1234 abcxyz456 เป็นต้น

```
>>> regex = r"^{[0-9]\{4\}} [A-Za-z]*"
>>> print(re.search(regex, "1234 Aa"))
<_sre.SRE_Match object; span=(0, 7), match='1234 Aa'>
>>> print(re.search(regex, "1234 AZaz"))
<_sre.SRE_Match object; span=(0, 9), match='1234 AZaz'>
>>> print(re.search(regex, "1234 abcxyz456"))
<_sre.SRE_Match object; span=(0, 11), match='1234
abcxyz'>
```

$regex = r"^{[0-9]\{4,5\}} [A-Z][a-z]\{2,\}"$ คือ ขึ้นต้น (^) ด้วยตัวเลข 0 ถึง 9 จำนวน 4 ถึง 5 ตัว ($[0-9]\{4,5\}$) \Rightarrow ตามด้วยอักษรว่าง 1 ตัว (" ") \Rightarrow ตามด้วยตัวอักษรตัวใหญ่ 1 ตัว ($[A-Z]$) \Rightarrow ตามด้วยอักษรตัวเล็กตั้งแต่ 2 ถึง N ตัว ($[a-z]\{2,\}$) เช่น 0123 Axyz, 0123 Aabcxyz, 0123 Zabcdefghijkl, 0123 Zabcdefghijkl%@

```
>>> regex = r"^{[0-9]\{4,5\}} [A-Z] [a-z]\{2,\}"
>>> print(re.search(regex, "0123 Axyz"))
<_sre.SRE_Match object; span=(0, 9), match='0123 Axyz'>
>>> print(re.search(regex, "0123 Aabcxyz"))
<_sre.SRE_Match object; span=(0, 12), match='0123
Aabcxyz'>
```

```
>>> print(re.search(regex, "0123 Zabcdefghijkl"))
<_sre.SRE_Match object; span=(0, 18), match='0123
Zabcdefghijkl'>
```

```
>>> print(re.search(regex, "0123 Zabcdefghijkl%@"))
<_sre.SRE_Match object; span=(0, 18), match='0123
Zabcdefghijkl'>
```

- | หมายถึง เสนอทางเลือกอย่างใดอย่างหนึ่ง เช่น abc|xyz แมตช์กับ abc หรือ xyz ก็ได้ หรือ (a|bc)de แมตช์กับ ade หรือ bcde เป็นต้น ดังตัวอย่างโปรแกรมต่อไปนี้

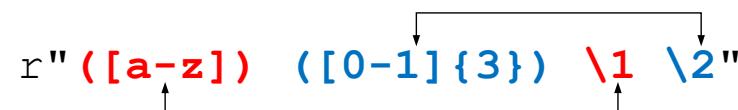
```
>>> import re
>>> regex = r"abc|xyz"
>>> print(re.search(regex, "abc"))
<_sre.SRE_Match object; span=(0, 3), match='abc'>
>>> print(re.search(regex, "xyz"))
<_sre.SRE_Match object; span=(0, 3), match='xyz'>
```

```
>>> regex = r"(a|bc)de"
>>> print(re.search(regex, "ade"))
<_sre.SRE_Match object; span=(0, 3), match='ade'>
>>> print(re.search(regex, "bcde"))
<_sre.SRE_Match object; span=(0, 4), match='bcde'>
```

- \$ แมตช์กับอักษรหรือคำที่เป็นคำลงท้ายของข้อความที่นำมาตรวจสอบเช่น abc\$ คือ abcabc, 123abc, abc\$3sabc เป็นต้น เช่น

```
>>> import re
>>> regex = r"abc$"
>>> print(re.search(regex, "123abc"))
<_sre.SRE_Match object; span=(3, 6), match='abc'>
>>> print(re.search(regex, "abc"))
<_sre.SRE_Match object; span=(3, 6), match='abc'>
>>> print(re.search(regex, "abc123"))
None
```

- \1...\9 แมตช์กับผลลัพธ์ที่เกิดจากกลุ่มของนิพจน์ปகติย่อย กลุ่มย่อยที่ 1 (\1), กลุ่มย่อยที่ 2 (\2), ..., กลุ่มย่อยที่ 9 (\9) ตามลำดับ เช่น regex = r"([a-z]) ([0-9]{3}) \1 \2" ผลลัพธ์ที่เกิดจาก \1 มีค่าเท่ากับ ([a-z]) และ \2 มีค่าเท่ากับ ([0-9]{3}) เป็นต้น



 $r"([a-z]) ([0-9]\{3\}) \1 \2"$

```
>>> import re
>>> regex = r"([a-z]) ([0-9]\{3\}) \1 \2"
>>> matchObj = re.search(regex, "a 123 a 123")
>>> print(matchObj)
<_sre.SRE_Match object; span=(0, 11), match='a 123 a
123'>
```

```
>>> print(matchObj.group(1))
a
>>> print(matchObj.group(2))
123
```



Note: กลุ่มของนิพจน์ปกติอย่างที่ต้องอยู่ภายในเครื่องหมาย (...) จากตัวอย่างด้านบน เช่น ([a-z]) คือ กลุ่มของนิพจน์ปกติอย่างที่ 1 ([1], [0-9]{3}) คือ กลุ่มของนิพจน์ปกติอย่างที่ 2 (2)

4. การค้นหาคำที่ดำเนินการเริ่มต้นและสิ้นสุดของข้อความ

จากที่ได้กล่าวมาแล้วว่าเมธอด search() จะค้นหาข้อความที่อยู่ตำแหน่งใดๆ ก็ได้ในข้อความ หรือสตริง เพื่อความสะดวกในการค้นหาคำที่อยู่ตำแหน่งเริ่มต้นหรือสิ้นสุดของข้อความ ไฟรอนได้เตรียมเมธอดชื่อว่า match() เพื่อใช้สำหรับตรวจสอบข้อความที่อยู่เฉพาะในตำแหน่งเริ่มต้นหรือสิ้นสุดของข้อความ เช่น

```
import re
s1 = "Python is the popular programming language."
s2 = "Everyone surely can write the Python language."
print(re.search(r"Python", s1))
print(re.search(r"Python", s2))
print(re.match(r"Python", s1))
print(re.match(r"Python", s2))
```



OUTPUT

```
<_sre.SRE_Match object; span=(0, 6), match='Python'>
<_sre.SRE_Match object; span=(30, 36), match='Python'>
<_sre.SRE_Match object; span=(0, 6), match='Python'>
None
```

เมธอด match() เป็นเมธอดพิเศษที่ใช้สำหรับค้นหาคำที่ดำเนินการเริ่มต้นของข้อความหรือสตริง ผู้เขียนโปรแกรมสามารถใช้สัญลักษณ์ ^ (Caret) นำหน้าคำที่ต้องการค้นหาร่วมกับเมธอด search() เพื่อทดสอบการใช้งานเมธอด match() ได้ หรือใช้ ^ กับข้อความที่ประกอบขึ้นจากข้อความหลายๆ บรรทัดเรียงต่อกันได้ (MULTILINE) โดยข้อความที่เป็น MULTILINE จะมี \n เชื่อมระหว่างแต่ละบรรทัด ซึ่งเมธอด match() ไม่สนับสนุนการทำงานกับ MULTILINE เช่น

```
import re
s1 = "Python is the popular programming language.\n"
s2 = "Everyone surely can write the Python language.\n"
print(re.search(r"^\Python", s1))
print(re.search(r"^\Python", s2))
```



OUTPUT

```
<_sre.SRE_Match object; span=(0, 6), match='Python'>
None
```

ทดสอบการใช้ ^ อีกรัง โดยการเชื่อมสตริง s1 และ s2 เข้าด้วยกัน ดังนี้

```
s = s2 + '\n' + s1 = "Everyone surely can write the Python
language.\nPython is the popular programming language.\n"
```



Note: โดยปกติ \n จะไม่ปรากฏให้เห็นที่ห้ายของข้อความ แต่ในทางปฏิบัติรหัสตั้งกล่าวจะอยู่ต่อห้ายข้อความเสมอ ในตัวอย่างข้างบน ผู้เขียนจึงเขียน \n ต่อห้ายข้อความ เพื่อแสดงให้ผู้อ่านเห็นว่ามีรหัส \n อยู่ด้วย

ผลจากการเชื่อมสตริง s2 กับ s1 เข้าด้วยกัน ส่งผลให้ \n เป็นคำแรกของสตริง s1 จากนั้นทำการทดสอบ โดยการใช้ ^ เพื่อค้นหาคำว่า 'Python' ใน s อีกรัง

```
s = s2 + '\n' + s1
print(re.search(r"^\Python", s))
```



None

จากผลลัพธ์ที่ได้คือ None แสดงให้เห็นว่า ^ มีความสามารถตรวจสอบคำที่อยู่ในตำแหน่งเริ่มต้นของสตริงเท่านั้น (ซึ่งคำแรกใน s คือคำว่า Everyone) แต่ถ้าต้องการให้ ^ สามารถค้นหาคำที่ต้องการจากสตริงที่เชื่อมต่อกัน ต้องระบุในเมธอด search() ด้วยโหมด MULTILINE (re.M) ดังนี้

```
import re
s1 = "Python is the popular programming language."
s2 = "Everyone surely can write the Python language."
s = s2 + '\n' + s1
print(re.search(r"^\Python", s, re.MULTILINE))
print(re.search(r"^\Python", s, re.M))
print(re.match(r"^\Python", s, re.M))
```



```
<_sre.SRE_Match object; span=(47, 53), match='Python'>
<_sre.SRE_Match object; span=(47, 53), match='Python'>
None
```

จากผลลัพธ์ที่ได้แสดงให้เห็นว่า ^ ต้องทำงานร่วมกับการค้นหาในโหมด MULTILINE สำหรับสตริงที่มีหลาย 줄 บรรทัด โดยเชื่อมต่อด้วย \n แต่สำหรับเมธอด match() ไม่สามารถใช้งานในโหมด MULTILINE ได้

จากตัวอย่างที่ผ่านมาเป็นการค้นหาคำที่ตำแหน่งเริ่มต้นของข้อความ สำหรับการค้นหาคำที่ตำแหน่งสุดท้ายของข้อความจะใช้สัญลักษณ์ '\$' (dollar sign) แทน ซึ่งการทำงานของสัญลักษณ์ ดังกล่าวจะแมตช์กับคำที่อยู่ท้ายข้อความในตำแหน่งก่อน \n เช่น abc\$ คือ abcabc, 123abc, abc\$3sabc เป็นต้น และสามารถใช้ '\$' ร่วมกับโหมด MULTILINE ได้ เช่น

```
import re
print(re.search(r"Python.$", "I like Python."))
print(re.search(r"Python.$", "I like Python and Perl."))
print(re.search(r"Python.$", "I like Python.\nSome prefer
Java or Perl."))
print(re.search(r"Python.$", "I like Python.\nSome prefer
Java or Perl.", re.M))
```



OUTPUT

```
<_sre.SRE_Match object; span=(7, 14), match='Python.'>
None
None
<_sre.SRE_Match object; span=(7, 14), match='Python.'>
```

จากตัวอย่างด้านบน Regex = r"Python.\$" และถึงโปรแกรมจะแมตช์กับข้อความที่ลงท้ายด้วยคำว่า "Python." เท่านั้น แต่ถ้าข้อความมีมากกว่า 1 บรรทัด (MULTILINE) ต้องกำหนดให้เมธอด search() ทำงานในโหมด MULTILINE โปรแกรมจึงสามารถค้นหาข้อมูลได้อย่างถูกต้อง

5. เมธอด re.search()

จากที่ได้กล่าวมาแล้วเบื้องต้นว่าเมธอด re.search() เป็นเมธอดสำคัญที่ใช้สำหรับค้นหาอักษร หรือคำในข้อความหรือสตริง ซึ่งเมธอดดังกล่าวมีรายละเอียดการใช้งานเพิ่มเติมดังนี้

รูปแบบของเมธอด re.search()

```
re.search(pattern, string, flags=0)
```

โดย pattern คือ นิพจน์ปกติหรือ regex ที่ใช้สำหรับตรวจสอบ, string คือ ข้อความหรือสตริงที่จะถูกตรวจสอบ, flags เป็นอ็อปชัน เพื่อใช้กำหนดเงื่อนไขพิเศษเพิ่มเติมในการเปรียบเทียบ ดังตารางที่ 12.1

ตารางที่ 12.1 flags

Flags	คำอธิบาย
re.I	ประมวลผลตัวอักษรแบบ case-insensitive (ตัวเล็กและใหญ่ต่างกัน) เช่น Python กับ python จะแตกต่างกัน
re.L	ประมวลผลตัวอักษรด้วยการอ้างอิงกับ Current locale
re.M	ประมวลผลกับข้อความที่มีจำนวนบรรทัดมากกว่า 1 บรรทัดขึ้นไป
re.S	ประมวลผลตัวอักษรระดับ ก็ได้ รวมถึงอักษรที่ใช้ขึ้นบรรทัดใหม่ด้วย (\n หรือ newline)
re.U	ประมวลผลกับตัวอักษรประเภทUnicode

เมธอด re.search() จะส่งค่ากลับเป็นอ็อปเจกต์เมื่อสามารถค้นหา (Match) คำหรือกลุ่มคำในข้อความ และจะส่งค่ากลับเป็น None เมื่อค้นหาคำหรือกลุ่มคำไม่พบ (Mismatch)

ผู้เขียนโปรแกรมสามารถดูรายละเอียดต่างๆ ของอ็อปเจกต์ที่ส่งกลับมาโดยใช้เมธอด group(), span(), start() และ end() ดังนี้

Program Example 12.2: re.search() method and other

```
1 #More details about re.search() method
2 import re
```

```

⇒3 regex = "[0-9]+"
⇒4 text = "Customer number: 232454, Date: February 12, 2011"
⇒5 matchObj = re.search(regex, text)
⇒6 print(matchObj)
⇒7 print("group->",matchObj.group())
⇒8 print("span->",matchObj.span())
⇒9 print("start->",matchObj.start())
⇒10 print("end->",matchObj.end())
⇒11 print("span[0]->",matchObj.span()[0])
⇒12 print("span[1]->",matchObj.span()[1])

```



OUTPUT

```

<_sre.SRE_Match object; span=(17, 23), match='232454'>
group->232454
span->(17, 23)
start->17
end->23
span[0]->17
span[1]->23

```

จากโปรแกรมตัวอย่างที่ 12.2 แสดงรายละเอียดเกี่ยวกับการใช้งานเมธอด `re.search()` โดยเริ่มต้นในบรรทัดที่ 3 โปรแกรมสร้าง `regex` เท่ากับ "[0-9]+" ซึ่งแมตช์กับตัวเลข 0 ถึง 9 และมีจำนวนไม่น้อยกว่า 1 ตัวขึ้นไป บรรทัดที่ 4 เป็นข้อความที่นำมาตรวจสอบ บรรทัดที่ 5 โปรแกรมเรียกใช้งานเมธอด `search()` โดยมีพารามิเตอร์ 2 ตัวคือ `regex` และ `text` ถ้าเมธอดดังกล่าวสามารถค้นหาคำที่กำหนดไว้ในตัวแปร `regex` พบริ候ข้อความที่เก็บในตัวแปร `text` เจอ เมธอด `search()` จะส่งอ้อมไปเจกต์กลับมายังผู้เรียก โดยเก็บไว้ในตัวแปรชื่อ `matchObj` แต่ถ้าค้นหาไม่พบ เมธอด `search()` จะส่งค่ากลับคืนมาเป็น `None` แทน

บรรทัดที่ 6 โปรแกรมทดสอบพิมพ์ข้อมูลในอ้อมไปเจกต์ที่ส่งกลับมาจากเมธอด `re.search()` ซึ่งข้อมูลที่แสดงออกมานัดหนึ่นคือ

`_sre.SRE_Match object` คือ ชื่อของโมดูลที่ใช้ในการค้นหาข้อมูล

`span=(17, 23)` คือ โมดูลคันพบคำที่ต้องการค้นหา โดยค่าดังกล่าวเริ่มต้นจากตำแหน่งที่ 17 ถึงตำแหน่งที่ 23 เป็นตำแหน่งสุดท้ายของข้อมูล (ข้อมูลจริงจะอยู่ที่ตำแหน่ง 17 - 22)

`match='232454'` คือคำที่ค้นพบ

บรรทัดที่ 7 โปรแกรมเรียกใช้งานเมธอด `group()` ซึ่งเก็บคำหรือกลุ่มคำที่ค้นพบ (ในตัวอย่างนี้มีเพียงคำเดียวคือ 232454) บรรทัดที่ 8 โปรแกรมเรียกใช้งานเมธอด `span()` ซึ่งเก็บตำแหน่งเริ่มต้นและสิ้นสุดของคำที่ค้นพบ (ในที่นี้คือตำแหน่งที่ 17 ถึง 23) บรรทัดที่ 9 โปรแกรมเรียกใช้งานเมธอด `start()` คือตำแหน่งเริ่มต้นของคำที่ค้นพบ (ตำแหน่งที่ 17) และบรรทัดที่ 10 โปรแกรมเรียกเมธอด `stop()` คือตำแหน่งสุดท้ายของคำที่ค้นพบ (ตำแหน่งที่ 23) ผู้เขียนโปรแกรมสามารถเรียกดูตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายของคำที่ค้นพบได้อีกวิธีหนึ่งคือ ใช้เมธอด `span()[0]` คือตำแหน่งเริ่มต้นของคำ และ

`span()[1]` คือตำแหน่งสุดท้ายของคำ ซึ่งคำสั่งดังกล่าวจะให้ผลลัพธ์เช่นเดียวกับการเรียกใช้เมธอด `start()` และ `stop()`

ค่าที่ส่งกลับจากเมธอด `group()` จะเป็นตัวแปรชนิดทัพเพิล ซึ่งมีความสามารถเก็บคำที่คันப์ได้มากกว่า 1 คำ ผู้เขียนโปรแกรมสามารถแสดงคำที่คันหาได้โดยใช้รูปแบบคำสั่งคือ `group([subgroup1, subgroup2, ..., subgroup])` ดังโปรแกรมตัวอย่างที่ 12.3

Program Example 12.3: `re.group([subgroup_id])` method

```

1  #More details about re.group() method
2  import re
3  ⇒  regex = "([0-9]+) .*: (.*)"
4  ⇒  text = "Customer number: 232454, Date: February 12, 2011"
5  ⇒  matchObj = re.search(regex, text)
6  print(matchObj)
7  print(matchObj.group())
8  print(matchObj.group(0))
9  print(matchObj.group(1))
10 print(matchObj.group(2))
11 print(matchObj.group(1,2))
```



OUTPUT

```

<_sre.SRE_Match object; span=(17, 48), match='232454, Date:
February 12, 2011'>
('232454', 'February 12, 2011')
232454, Date: February 12, 2011
232454
February 12, 2011
('232454', 'February 12, 2011')
```

จากโปรแกรมตัวอย่างที่ 12.3 แสดงการใช้งานเมธอด `re.group()` เพื่อแสดงกลุ่มคำที่คันப์โดยในบรรทัดที่ 3 โปรแกรมประกาศนิพจน์ปกติให้กับตัวแปร `regex` เท่ากับ "[0-9]+).*: (.*)" หมายถึงคำที่ต้องการค้นหาเริ่มต้นด้วยตัวเลข 0 ถึง 9 และมีจำนวนตัวเลขอย่างน้อย 1 ตัวขึ้นไป ([0-9]+) ⇒ ตามด้วยอักษรใดๆ ก็ได้แบบไม่จำกัดความยาว (.*) ⇒ ตามด้วยเครื่องหมาย : ⇒ ตามด้วยตัวอักษร ว่าง 1 ตัว (" ") ⇒ ตามด้วยอักษรใดๆ ก็ได้แบบไม่จำกัดความยาว (.*) ปิดท้ายคำที่ต้องการค้นหาบรรทัดที่ 4 เป็นข้อความที่ต้องการค้นหา บรรทัดที่ 5 โปรแกรมเรียกใช้คำสั่ง `re.search()` โดยมีพารามิเตอร์ 2 ตัวคือ `regex` และ `text` อ้อปเจ็กต์ที่ส่งกลับจากเมธอด `re.search()` จะเก็บไว้ในตัวแปร `matchObj` บรรทัดที่ 6 โปรแกรมพิมพ์ค่าต่างๆ ในอ้อปเจ็กต์ `matchObj` ที่ส่งกลับมาจากเมธอด `re.search()`

บรรทัดที่ 7 โปรแกรมสั่งพิมพ์กลุ่มคำย่ออยู่ที่คันப์ทั้งหมดออกจากภาพ ผลลัพธ์ที่ได้มี 2 กลุ่มคำย่อคือ "232454" และ "Date: February 12, 2011" บรรทัดที่ 8 สั่งพิมพ์กลุ่มคำย่อที่คันப์ทั้งหมดในตัวแปรทัพเพิลออกจากภาพ บรรทัดที่ 9 โปรแกรมสั่งพิมพ์กลุ่มคำย่อที่ 1 ด้วยการเรียกใช้เมธอด `group(1)` คือ "232454" บรรทัดที่ 10 เรียกใช้เมธอด `group(2)` เพื่อสั่งพิมพ์กลุ่มคำย่อที่ 2 คือ "Date:

February 12, 2011" และบรรทัดที่ 11 โปรแกรมสั่งพิมพ์กลุ่มคำย่ออยู่ที่คันபบทั้งสองคำอ กจากภาษาด้วยคำสั่ง group(1, 2) ดังตัวอย่างโปรแกรมที่ 12.3

ผู้เขียนโปรแกรมสามารถแสดงผลลัพธ์ย่อๆ ที่เกิดจาก regex (group()) ได้มากกว่า 1 จำนวนเช่น

Program Example 12.4: `re.group([subgroup_id]) method`

```

1  #Another example for re.group(index) method
2  import re
3  matchObj = re.match("(Hello|Hi) (Tom|Thom) (Bombadil|Jason)",
4      "Hello Tom Bombadil")
5  if not matchObj is None:
6      print (matchObj.group(0))
7      print (matchObj.group(1))
8      print (matchObj.group(2))
9      print (matchObj.group(3))

```



OUTPUT

```

Hello Tom Bombadil
Hello
Tom
Bombadil

```

โปรแกรมตัวอย่างที่ 12.4 เป็นตัวอย่างการใช้งานเมธอด group() อีกด้วยหนึ่ง เริ่มต้นในบรรทัดที่ 3 โปรแกรมสร้างนิพจน์ปกติคือ "(Hello|Hi) (Tom|Thom) (Bombadil|Jason)" หมายถึง คันหาคำที่ขึ้นต้นด้วยคำว่า Hello หรือ Hi ก็ได้ \Rightarrow ตามด้วยคำว่า Tom หรือ Thom ก็ได้ \Rightarrow ตามด้วยคำว่า Bomdadil หรือ Jason ตามลำดับ สำหรับข้อความที่ใช้สำหรับทดสอบคือ "Hello Tom Bombadil" โดยเปลี่ยนจากการเรียกเมธอด re.search() มาเป็นเมธอด re.match() แทน ผลลัพธ์ที่ได้จากการตรวจสอบจะเก็บในตัวแปร matchObj

ผลลัพธ์จะถูกตรวจสอบด้วยคำสั่ง if (บรรทัดที่ 4) เมื่อเมธอด match() สามารถค้นหาคำที่ต้องการพบ โปรแกรมจะพิมพ์กลุ่มคำย่ออยู่ที่คันபบโดยเรียงตามลำดับของนิพจน์ปกติที่ระบุ เช่น เมื่อคำได้คำหนึ่งระหว่าง Hello กับ Hi ถูกแมตช์ (บรรทัดที่ 6) ผลลัพธ์ที่ได้จะเก็บไว้ใน matchObj.group(1), ถ้าคำได้คำหนึ่งระหว่าง Tom กับ Thom ถูกแมตช์ (บรรทัดที่ 7) ผลลัพธ์ที่ได้จะเก็บไว้ใน matchObj.group(2) และถ้าคำได้คำหนึ่งระหว่าง Bombadil กับ Jason ถูกแมตช์ (บรรทัดที่ 8) ผลลัพธ์ที่ได้จะเก็บไว้ใน matchObj.group(3) สำหรับ matchObj.group(0) จะเก็บผลลัพธ์ของกลุ่มคำย่ออยู่ทั้งหมดไว้ ผลลัพธ์ของโปรแกรมแสดงดังในตัวอย่าง OUTPUT ด้านบน

สำหรับตัวอย่างการใช้งานจริง เช่น การจัดการกับแฟ้ม XML หรือ HTML ซึ่งมีรูปแบบเนื้อหาของแฟ้มดังนี้ (แฟ้มชื่อ tags.xml)

```
<title>Programming Fundamentals and Python</title>
```

```
<author>Steven Bird, Ewan Klein, Edward Loper</author>
```

<city>Pennsylvania, USA</city>

เมื่อผู้เขียนโปรแกรมต้องการจัดเรียงข้อมูลในแฟ้ม tags.xml ใหม่ให้มีรูปแบบดังต่อไปนี้คือ

title: Programming Fundamentals and Python

author: Steven Bird, Ewan Klein, Edward Loper

city: Pennsylvania, USA

สามารถเขียน regex เพื่อจัดเรียงข้อมูลได้ดังนี้ regex = r"<([a-z]+)>(.*)</\1>" ดังแสดงในโปรแกรมที่ 12.5



Input File: tags.xml

```
<title>Programming Fundamentals and Python</title>
<author>Steven Bird, Ewan Klein, Edward Loper</author>
<city>Pennsylvania, USA</city>
```

Program Example 12.5: applied regex for XML

```
1 #XML example
2 import re
3 f = open("tags.xml")
4 for i in f:
5     res = re.search(r"<([a-z]+)>(.*)</\1>", i)
6     print(res.group(1) + ":" + res.group(2))
7 f.close()
```



OUTPUT title: Programming Fundamentals and Python
author: Steven Bird, Ewan Klein, Edward Loper
city: Pennsylvania, USA

จากตัวอย่างโปรแกรมที่ 12.5 แสดงตัวอย่างการใช้ regex กับแฟ้มข้อมูล xml เพื่ออ่านและจัดเรียงข้อมูลตามที่ผู้เขียนโปรแกรมต้องการ เริ่มต้นในบรรทัดที่ 3 โปรแกรมทำการเปิดแฟ้มข้อมูลชื่อ tags.xml โดยเนื้อหาของข้อมูลดังแสดงใน Input File ด้านบน แฟ้มที่ถูกเปิดแล้วจะอ้างอิงด้วยอ้อม��เก็ต์ f จากนั้นโปรแกรมจะอ่านข้อมูลจากแฟ้มเข้ามาครั้งละ 1 บรรทัด โดยใช้คำสั่ง for (บรรทัดที่ 4) ข้อมูลจากแฟ้มในแต่ละบรรทัดที่อ่านได้ จะถูกเปรียบเทียบกับ regex = r"<([a-z]+)>(.*)</\1>" ซึ่งหมายถึง ข้อความต้องเป็นตัวอักษรตัวเล็กที่มีจำนวนมากกว่า 1 ตัวอักษรขึ้นไปและอยู่ภายใต้เครื่องหมาย <> (เกิดจากกลุ่มของอักษรที่ประกอบไปด้วย <([a-z]+)>) เรียกว่า regex กลุ่มย่อยที่ 1 \Rightarrow ตามด้วยตัวอักษรใดๆ ก็ได้ (รวมไปถึงไม่มีตัวอักษรด้วย) แบบไม่จำกัดจำนวน (เกิดจาก (.*)) \Rightarrow ตามด้วยนิพจน์ปกติอย่างกลุ่มที่ 2 ซึ่งเหมือนกับ regex กลุ่มย่อยที่ 1 (\1) โดยต้องขึ้นต้นด้วยสัญลักษณ์ '/' และอยู่ภายใต้เครื่องหมาย <> ผลลัพธ์ที่เกิดจากนิพจน์ปกติ r"<([a-z]+)>(.*)</\1>" จะมี 2 ค่า (สังเกตผลลัพธ์

จะเกิดขึ้นภายใต้เครื่องหมาย "(...)" คือ title และ "Programming Fundamentals and Python" สำหรับบรรทัดที่ 1 ในไฟล์ tags.xml

บรรทัดที่ 6 โปรแกรมทำการเชื่อมข้อความที่เป็นผลลัพธ์จากนิพจน์ `r"<([a-z]+)>(.**)</1>"` ด้วยเครื่องหมาย ":" และพิมพ์ออกจากการ โปรแกรมจะทำการวนซ้ำอ่านค่าจากไฟล์ tags.xml ไปเรื่อยๆ จนหมดไฟล์ เมื่อหมดไฟล์แล้วโปรแกรมจะปิดไฟล์ด้วยเมธอด `close()` ผลลัพธ์ที่ได้แสดงในตัวอย่าง OUTPUT ด้านบน

ตัวอย่างที่ 12.6 แสดงตัวอย่างการค้นหาหมายเลขโทรศัพท์จากสมุดโทรศัพท์

Program Example 12.6: applied regex with phone book

```

1 #Phone book
2 import re
3 ⇒3 phone_list = ["555-8396 Neu, Allison", "Burns, C.
Montgomery", "555-5299 Putz, Lionel", "555-7334 Simpson,
Homer Jay"]
4 ⇒4 for i in phone_list:
5 ⇒5     res = re.search(r"([0-9-]*)\s*([A-Za-z]+),\s+(.*")", i)
6 ⇒6     print(res.group(3) + " " + res.group(2) + " "
res.group(1))

```



Allison Neu 555-8396
C. Montgomery Burns
Lionel Putz 555-5299
Homer Jay Simpson 555-7334

OUTPUT

จากโปรแกรมตัวอย่างที่ 12.6 แสดงการประยุกต์ใช้งาน regex กับการค้นหาหมายเลขที่อยู่โทรศัพท์ โดยเริ่มต้นบรรทัดที่ 3 โปรแกรมสร้างตัวแปรชนิดลิสต์ชื่อ `phone_list` สำหรับเก็บชื่อผู้ใช้งานและหมายเลขโทรศัพท์ บรรทัดที่ 4 โปรแกรมอ่านข้อมูลจากตัวแปร `phone_list` มาครั้งละ 1 บรรทัดเก็บไว้ในตัวแปร `i` ต่อจากนั้นโปรแกรมจะนำค่าที่อยู่ในตัวแปร `i` มาเปรียบเทียบกับ regex ที่มีนิพจน์เป็น `r"([0-9-]*)\s*([A-Za-z]+),\s+(.*")` หมายถึง หมายเลขโทรศัพท์ที่ขึ้นต้นด้วย 0 ถึง 9 หรือ "-" และมีความยาวกี่ตัวอักษรก็ได้ $([0-9-]*)$ \Rightarrow ตามด้วยช่องว่างกี่ตัวก็ได้ $(\s*)$ \Rightarrow ตามด้วย ตัวอักษรตัวใหญ่หรือเล็กก็ได้มากกว่า 1 ตัวอักษรขึ้นไป $([A-Za-z]+)$ \Rightarrow ตามด้วย "," \Rightarrow ตามด้วยช่องว่างอย่างน้อย 1 ตัว $(\s+)$ \Rightarrow ตามด้วยตัวอักษรใดๆ ก็ได้แบบไม่จำกัดจำนวน $((.*))$ ปิดท้าย

ผลลัพธ์ที่ได้มี 3 ค่า (สังเกตที่เครื่องหมาย (...) หมายถึงนิพจน์ปกติย่อ) โดยค่าที่ 1 เก็บไว้ใน `res.group(1)` ซึ่งเกิดจากกลุ่ม regex ย่อที่ 1 $([0-9-]*),$ ค่าที่ 2 เก็บไว้ใน `res.group(2)` ซึ่งเกิดจากกลุ่ม regex ย่อที่ 2 $([A-Za-z]+)$ และค่าที่ 3 เก็บไว้ใน `res.group(3)` ซึ่งเกิดจากกลุ่ม regex ย่อที่ 3 $(.*")$ ผลลัพธ์ที่ได้จะถูกเชื่อมด้วยสัญลักษณ์ " " และพิมพ์ผลลัพธ์ที่ได้ออกทางจอภาพ (ในบรรทัดที่ 6)

6. การค้นหาและแทนที่คำด้วยเมธอด re.sub()

ในโมดูล re มีเมธอดสำคัญเพื่อทำหน้าที่แทนที่ข้อมูลคือ sub โดยมีรูปแบบคือ

re.sub(pattern, repl, string, max=0)

โดย pattern คือ นิพจน์ปกติหรือ regex ที่ใช้สำหรับตรวจสอบ, repl คือ ข้อความที่ต้องการแทนที่, string คือ ข้อความหรือสตริงที่จะถูกตรวจสอบ, max เป็นอ็อปชัน สำหรับใช้กำหนดจำนวนข้อความที่จะแทนที่ ถ้าไม่มีการกำหนดค่าให้กับตัวพารามิเตอร์ max ข้อความจะถูกแทนที่ทั้งหมด ตัวอย่างที่ 12.7 แสดงการใช้งานเมธอด re.sub()

Program Example 12.7: re.sub()

```

1 # re.sub() testing
2 import re
⇒3 phone = "2004-959-559 # This is Phone Number"
⇒4 # Delete Python-style comments
⇒5 num = re.sub(r'#.*$', "", phone)
⇒6 print ("Phone Num : ", num)
⇒7 # Remove anything other than digits
⇒8 num = re.sub(r'\D', "", phone)
⇒9 print ("Phone Num : ", num)

```



Phone Num : 2004-959-559
Phone Num : 2004959559

จากตัวอย่างโปรแกรมที่ 12.7 แสดงการใช้งานเมธอด re.sub() ในการแทนค่าข้อมูลตามที่ผู้ใช้ต้องการด้วย regex ที่มีค่าเท่ากับ '#.*\$' เริ่มต้นในบรรทัดที่ 3 โปรแกรมทำการປະກາດตัวแปร phone มีค่าเท่ากับ "2004-959-559 # This is Phone Number" บรรทัดที่ 4 โปรแกรมทำการลบข้อความ "# This is Phone Number (comment)" ออกจากข้อมูลในตัวแปร phone โดยใช้อักษร "" แทนที่

ถัดมาในบรรทัดที่ 5 โปรแกรมทำการพิมพ์ข้อความที่ลับ comment ออกแล้ว ออกจากภาพ บรรทัดที่ 7 โปรแกรมทำการลบอักษรที่ไม่ใช่ตัวเลขออกจากข้อความ ('\D') ทำให้ผลลัพธ์สุดท้ายเหลือเฉพาะตัวเลขเท่านั้น ดังตัวอย่างข้างบน

7. เมธอด re.findall()

ในโมดูล re มีเมธอดที่นำเสนอไว้อีกเมธอดคือ findall() เมธอดดังกล่าวนี้จะค้นหาคำทั้งหมดในข้อความที่ต้องการตรวจสอบทั้งหมด ไม่ว่าข้อความนั้นๆ จะมีความยาวเท่าใดก็ได้ แตกต่างจากเมธอด re.search() คือ ซึ่งจะค้นหาเฉพาะคำแรกที่ค้นพบเท่านั้น แต่สำหรับเมธอด findall() จะค้นหาหมดทั้งแฟ้ม คำที่ส่งกลับจะเป็นรายการคำที่ค้นพบทั้งหมดและเป็นตัวแปรชนิดลิสต์ รูปแบบของเมธอด findall() คือ

re.findall(pattern, string)

โดย pattern คือ นิพจน์ปกติหรือ regex ที่ใช้สำหรับตรวจสอบ, string คือ ข้อความหรือสตริงที่จะถูกตรวจสอบ ซึ่งแสดงการใช้งานดังต่ออย่างโปรแกรมที่ 12.8

Program Example 12.8: re.findall()

```

1 # re.findall() testing
2 import re
3 value = "abc 123 def 456 dot map pat"
# Find all words starting with d or p.
4 list = re.findall("[dp]\w+", value)
5 # Print result.
6 print(list)

```


OUTPUT

['def', 'dot', 'pat']

จากตัวอย่างโปรแกรมที่ 12.8 แสดงการใช้เมธอด `findall()` ในการค้นหาคำที่ต้องการ บรรทัดที่ 3 โปรแกรมกำหนดค่า `value` เท่ากับ "abc 123 def 456 dot map pat" เพื่อใช้สำหรับเป็นข้อความ ตัวนับที่ต้องการค้นหา บรรทัดที่ 4 โปรแกรมเรียกใช้งานเมธอด `findall()` ซึ่งมีพารามิเตอร์ 2 ตัวคือ `regex` เท่ากับ "[dp]\w+" หมายถึง คำที่ขึ้นต้นด้วยตัวอักษร 'd' หรือ 'p' ([dp]) \Rightarrow ตามด้วยตัวอักษร อักษรใดไม่ต่ำกว่า 1 ตัว (\w+) และ `value` คือข้อความที่ต้องการตรวจสอบ บรรทัดที่ 6 โปรแกรมสั่งพิมพ์ผลลัพธ์ที่ค้นหาได้ออกจากภาพ

เพื่อให้เข้าใจการทำงานของเมธอด `findall()` เพิ่มเติม ผู้เขียนจะอยกตัวอย่างการใช้เมธอด `findall()` สำหรับค้นหาชื่ออีเมล ซึ่งมีรูปแบบดังนี้คือ

```
email_list = 'Sophia Emma@google.com, Isabella Olivia Ava@abc.com Emily Abigail'
```

Program Example 12.9: finding email address

```

1 # To find email address
2 import re
3 email_list = 'Sophia Emma@google.com, Isabella Olivia
Ava@abc.com Emily Abigail'
4
# re.findall() returns a list of all the found email strings
# ['Emma@google.com', 'Ava@abc.com']
5 emails = re.findall(r'[\w\.-]+@[\\w\.-]+', email_list)
6
7 for email in emails:
# do something with each found email string
8     print (email)
9
10

```


OUTPUT

Emma@google.com
Ava@abc.com

จากตัวอย่างโปรแกรมที่ 12.9 แสดงการค้นหาชื่ออีเมลที่ผู้ใช้ต้องการ เริ่มต้นบรรทัดที่ 3

โปรแกรมกำหนดชื่ออีเมลให้กับตัวแปร `email_list` มีค่าเท่ากับ '`Sophia Emma@google.com, Isabella Olivia Ava@abc.com Emily Abigail`' บรรทัดที่ 6 โปรแกรมเรียกใช้เมธอด `findall()` เพื่อค้นหาชื่ออีเมล ที่ต้องการ โดยเมธอดดังกล่าวต้องการพารามิเตอร์ 2 ตัวคือ `regex` มีค่าเท่ากับ `r'[\w\.-]+@[\\w\.-]+'` หมายถึง ชื่ออีเมลต้องขึ้นต้นด้วยตัวอักษรหรือสัญลักษณ์ '-' มีจำนวนไม่น้อยกว่า 1 ตัว (`[\w\.-]+`) \Rightarrow ตามด้วยเครื่องหมาย @ \Rightarrow ตามด้วยตัวอักษรหรือสัญลักษณ์ '-' มีจำนวนไม่น้อยกว่า 1 ตัวปิดท้ายชื่ออีเมล และพารามิเตอร์ตัวที่ 2 คือ รายชื่ออีเมลที่ต้องการค้นหา (`email_list`) ผลลัพธ์ที่ส่งกลับจากเมธอด `findall()` จะเป็นรายการชื่ออีเมลโดยเก็บอยู่ในตัวแปรชนิดลิสต์ (`emails`) บรรทัดที่ 8 โปรแกรมทำการอ่านค่าในตัวแปร `emails` ทีละชื่อ เก็บไว้ในตัวแปรชื่อ `email` และพิมพ์ข้อมูลดังกล่าวออกทางจอภาพ (บรรทัดที่ 10) ผลลัพธ์แสดงในตัวอย่าง OUTPUT ด้านบน

8. เมธอด `re.match()`

ในโมดูล `re` มีเมธอดชื่อ `match()` ซึ่งสามารถค้นหาคำในข้อความ เช่นเดียวกับเมธอด `search()` รูปแบบของเมธอด `match()` คือ

```
re.match(pattern, string, flags=0)
```

โดย `pattern` คือ นิพจน์ปกติหรือ `regex` ที่ใช้สำหรับตรวจสอบ, `string` คือ ข้อความหรือสตริงที่จะถูกตรวจสอบ และ `flags` เป็นอ้อปชัน เพื่อใช้กำหนดเงื่อนไขพิเศษเพิ่มเติมในการเปรียบเทียบ ดังตารางที่ 12.1 (สามารถใช้งานได้มากกว่า 1 อ้อปชัน โดยใช้ตัวดำเนินการ OR หรือ '|' เช่น `re.M|re.I`) ซึ่งแสดงการใช้งานดังตัวอย่างโปรแกรมที่ 12.10

Program Example 12.10: <code>re.match()</code> example	
---	--

```

1 import re
2 # Sample strings
3 list = ["dog dot", "do don't", "dumb-dumb", "no match"]
4 # Loop
5 for element in list:
6     # Match if two words starting with letter d
7     m = re.match("(d\w+) \W(d\w+)", element)
8     # See if success
9     if m:
          print(m.groups())

```



OUTPUT

```
('dog', 'dot')
('do', 'don')
('dumb', 'dumb')
```

จากตัวอย่างโปรแกรมที่ 12.10 แสดงการใช้งานเมธอด `match()` เพื่อค้นหาคำที่ต้องการ โดยบรรทัดที่ 3 โปรแกรมทำการประมวลผลรูปแบบ list มีค่าเท่ากับ `["dog dot", "do don't", "dumb-dumb", "no match"]` เพื่อใช้สำหรับเป็นข้อความตัวอย่างที่ต้องการค้นหา บรรทัดที่ 4 โปรแกรมทำการอ่านข้อมูลจากตัวแปร `list` เข้ามาประมวลผลครั้งละ 1 ข้อความ เก็บไว้ในตัวแปรชื่อ `element` ต่อจากนั้นโปรแกรมจะเรียกใช้เมธอด `match()` เพื่อค้นหาข้อความที่ต้องการ โดยเมธอดดังกล่าวมีพารามิเตอร์ 2 ตัวคือ `regex` มีค่าเท่ากับ `"(d\w+)\W(d\w+)"` หมายถึง ข้อความที่ต้องการค้นหาขึ้นต้นด้วยตัวอักษร 'd' ตัวเดียว \Rightarrow ตามด้วยตัวอักษรใดๆ ก็ได้ไม่น้อยกว่า 1 ตัวอักษร (`\w+`) \Rightarrow ตามด้วยตัวอักษร 'd' ตัวเดียว \Rightarrow ตามด้วยตัวอักษรใดๆ ก็ได้ไม่น้อยกว่า 1 ตัวอักษร (`\w+`) ปิดท้ายคำ และพารามิเตอร์ตัวที่ 2 คือ ข้อความที่ต้องการตรวจสอบ ผลลัพธ์ที่ได้จากการค้นหาจะเก็บไว้ในตัวแปร `m` และสั่งพิมพ์คำดังกล่าวออกทางจอภาพ (บรรทัดที่ 9) โดยใช้เมธอด `m.groups()` ซึ่งผลลัพธ์ของข้อมูลจะประกอบไปด้วย 2 ส่วน คือ ส่วนที่ 1 ผลลัพธ์ที่เกิดจากนิพจน์ปักติ `"(d\w+)"` และส่วนที่ 2 ผลลัพธ์ที่เกิดจากนิพจน์ปักติ `"(\d\w+)"` เช่นเดียวกัน ผู้เขียนโปรแกรมสามารถแสดงผลโดยใช้ `m.groups(1)` และ `m.groups(2)` ก็จะได้ผลลัพธ์เช่นเดียวกับการใช้คำสั่ง `m.groups()`

9. เมธอด `re.split()`

โมดูล `re` มีเมธอดชื่อ `split()` ซึ่งสามารถค้นหาและตัดคำออกจากข้อความได้แบบอัตโนมัติ มีรูปแบบของเมธอดคือ

`re.split(pattern, string)`

โดย `pattern` คือ นิพจน์ปักติหรือ `regex` ที่ใช้สำหรับตรวจสอบ, `string` คือ ข้อความหรือสตริงที่จะถูกตรวจสอบ ซึ่งแสดงการใช้งานดังตัวอย่างโปรแกรมที่ 12.11

Program Example 12.11: <code>re.split() example</code>	
1	<code>import re</code>
2	<code># Input string</code>
⇒3	<code>value = "one 1 two 2 three 3"</code>
	<code># Separate on one or more non-digit characters</code>
⇒4	<code>result = re.split("\D+", value)</code>
5	<code># Print results</code>
⇒6	<code>for element in result:</code>
7	<code> print(element)</code>



1
2
3

จากตัวอย่างโปรแกรมที่ 12.11 แสดงตัวอย่างการใช้งานเมธอด `split()` เพื่อใช้สำหรับตัดหรือแยกคำที่ต้องการจากข้อความที่ใช้ตรวจสอบ เริ่มต้นบรรทัดที่ 3 โปรแกรมกำหนดค่าให้กับตัวแปร `value` เท่ากับ "one 1 two 2 three 3" บรรทัดที่ 4 โปรแกรมเรียกใช้งานเมธอด `split()` เพื่อคัดแยกคำออกจากข้อความ โดยมีพารามิเตอร์ 2 ตัวคือ `regex` ที่ใช้สำหรับคัดแยกคำออกจากข้อความ ในตัวอย่างนี้ `regex` มีค่าเท่ากับ "`D+`" หมายถึง ตัวเลขใดๆ ตั้งแต่ 0 ถึง 9 โดยต้องมีจำนวนมากกว่า 1 ตัวขึ้นไป และพารามิเตอร์ตัวที่ 2 คือข้อความที่ต้องการทดสอบ ผลลัพธ์ที่ได้จากเมธอดดังกล่าว จะมีเฉพาะตัวเลขเท่านั้น และเก็บไว้ในตัวแปรชนิดลิสต์ชื่อ `result` ต่อจากนั้นในบรรทัดที่ 6 โปรแกรมใช้คำสั่ง `for` เพื่ออ่านคำมาจากการตัวแปร `result` และสั่งพิมพ์ที่จะคำนหนด ผลลัพธ์ที่ได้แสดงใน OUTPUT ด้านบน

10. การกำหนดชื่อให้กับ regex (Named group)

ในบางกรณี `regex` มีความซับซ้อนหรือมีความยาวมาก ๆ อาจจะทำให้ผู้เขียนโปรแกรมทำความเข้าใจได้ยาก ดังนั้นไฟรอนจึงได้จัดเตรียมคำสั่งที่ช่วยให้ผู้เขียนโปรแกรมสามารถจัดกลุ่มคำสั่งต่าง ๆ ที่ประกอบกันเข้าเป็น `regex` และตั้งชื่อใหม่สำหรับใช้เรียกเพื่อให้ความสะดวกในการใช้งาน เรียกว่า `Named groups` ซึ่งมีรูปแบบการใช้งานดังนี้ คือ

`?P<Named group>`

โดย `Named group` คือ ชื่อที่ใช้สำหรับเรียกชื่อกลุ่มของนิพจน์ปกติหรือ `regex` แสดงดังในตัวอย่างโปรแกรมที่ 12.12

Program Example 12.12: Named group example	
---	--

```

1 import re
2 # A string
3 name = "Clyde Griffiths"
4 # Match with named groups
5 m = re.match("(?P<FIRST>\w+) \w+ (?P<LAST>\w+)", name)
6 # Print groups using names as id
7 if m:
8     print(m.group("FIRST"))
9     print(m.group("LAST"))

```



Clyde
Griffiths

OUTPUT

จากตัวอย่างโปรแกรมที่ 12.12 แสดงการสร้าง `Named group` หรือการตั้งชื่อกลุ่มของ `regex` โดยเริ่มต้นในบรรทัดที่ 3 โปรแกรมกำหนดค่าให้กับตัวแปร `name` เท่ากับ "Clyde Griffiths" เพื่อใช้ในการทดสอบ บรรทัดที่ 4 โปรแกรมเรียกใช้งานเมธอด `match()` เพื่อค้นหาคำที่ต้องการ โดยมีพารามิเตอร์ 2 ตัวคือ `regex` มีค่าเท่ากับ "`(?P<FIRST>\w+)\w+ (?P<LAST>\w+)"` หมายถึง ต้องการ

ค้นหาคำที่ขึ้นต้นด้วยตัวอักษรใดๆ ก็ได้อย่างน้อย 1 ตัว (\w+) โดยเรียกชื่อกลุ่มของ regex ใหม่นี้ว่า "FIRST" เกิดจากนิพจน์ ?P<FIRST> \Rightarrow ค้นด้วยตัวอักษรที่ไม่ใช่ตัวอักษรอย่างน้อย 1 ตัว (\W+) \Rightarrow ปิดท้ายด้วยตัวอักษรใดๆ ก็ได้อย่างน้อย 1 ตัว และเรียกชื่อกลุ่ม regex ใหม่นี้ว่า "LAST" เกิดจากนิพจน์ ?P<LAST>

ผู้เขียนโปรแกรมสามารถเรียกดูผลลัพธ์โดยเรียกชื่อ FIRST และ LAST แทน (บรรทัดที่ 7 และ 8) ดังตัวอย่างข้างบน นอกจากการใช้ Named group แล้ว ไฟชอนยังเตรียมเมธอด groupdict() เพื่อ คำนวณความสะดวกให้กับผู้เขียนโปรแกรมสามารถเรียกดูผลลัพธ์ที่เกิดขึ้นจากการเรียกใช้งาน regex ด้วย ดังตัวอย่างโปรแกรมที่ 12.13

Program Example 12.13: `groupdict()` example

```

1 import re
2 name = "Clyde Griffiths"
3 # Match names
4 m = re.match("(?P<first>\w+)\W+(?P<last>\w+)", name)
5 if m:
6     # Get dict
7     d = m.groupdict()
8     # Loop over dictionary with for-loop
9     for t in d:
10         print(" key:", t)
11         print("value:", d[t])

```



key: first
value: Clyde
key: last
value: Griffiths

จากตัวอย่างโปรแกรมที่ 12.13 แสดงการใช้เมธอด groupdict() โดยตัวอย่างนี้จะทำงานเหมือนกับ โปรแกรมตัวอย่างที่ 12.12 แต่แตกต่างกันคือ ในตัวอย่างนี้ ผลลัพธ์ที่ได้จากการค้นหาจะถูกแปลงไปเป็น ข้อมูลประเภทดิกชันนารี (บรรทัดที่ 6) เมื่อต้องการแสดงผลลัพธ์ของการทำงานสามารถทำได้โดยเรียก ผ่านคีย์แทน (บรรทัดที่ 9 และ 10)

11. ตัวอย่างการเขียน regex พร้อมคำอธิบาย

นิพจน์ปกติ	คำอธิบาย
r"xell.*soft"	แมตช์กับคำที่ขึ้นต้นด้วย "xell" (xell) \Rightarrow ตามด้วยอักษรใดๆ ก็ได้ แบบไม่จำกัดความยาว (*) \Rightarrow ปิดท้ายด้วย "soft" เช่น xell123soft, xell\$%absoft, xell-@1-soft, xell_xell_soft เป็นต้น
r"xell(.*)soft"	เหมือนกับนิพจน์ r"xell.*soft" แต่นิพจน์ r"xell(.*)soft" มีการจัดกลุ่ม ย่อของนิพจน์ด้วยสัญลักษณ์ (...) ในตัวอย่างคือ (*)

r"xell\s{1,4}soft"	แมตช์กับคำที่ขึ้นต้นด้วย "xell" (xell) \Rightarrow ตามช่องว่าง แท็บ หรือขีนบรรทัดใหม่ (\n) จำนวน 1 ถึง 4 ตัว {1,4} \Rightarrow ปิดท้ายด้วย "soft" เช่น xell soft, xell absoft, xell\nsoft เป็นต้น
r"xell[0-9ABCDEF]*soft"	แมตช์กับคำที่ขึ้นต้นด้วย "xell" \Rightarrow ตามด้วยเลข 0 ถึง 9 หรืออักษรตัวใหญ่ A ถึง F แบบไม่จำกัดจำนวน ([0-9ABCDEF]*) \Rightarrow ปิดท้ายด้วย "soft" เช่น xell123Asoft, xelliABC12soft, xell1A2Bsoft เป็นต้น
r"hari (?and or) mary"	แมตช์กับคำที่ขึ้นต้นด้วย "hari" \Rightarrow ตามด้วยคำว่า "and" หรือ "or" (?and or) \Rightarrow ปิดท้ายด้วย "mary" เช่น harry and mary, harry or mary เป็นต้น โดย (?) หมายถึงไม่ใช่การจัดกลุ่มนิพจน์ปกติอยู่
r"harvex\d+"	แมตช์กับคำที่ขึ้นต้นด้วย "harvex" \Rightarrow ตามด้วยตัวเลขตั้งแต่ 1 ตัวขึ้นไป (\d+) เช่น harvex123, harvex45678 เป็นต้น
r"harvex\+?"	แมตช์กับคำที่ขึ้นต้นด้วย "harvex" \Rightarrow ตามด้วยเครื่องหมาย + หรือไม่มีก็ได้ (+?) เช่น harvex, harvex+ เป็นต้น โดย () หมายถึง หลีกเลี่ยง การประมวลผลเครื่องหมายพิเศษ ในกรณีนี้ + จะไม่ใช่การบวก แต่เป็นสัญลักษณ์ "+" แทน
r"harvex(\+\+)?"	แมตช์กับคำที่ขึ้นต้นด้วย "harvex" \Rightarrow ตามด้วยเครื่องหมาย ++ หรือไม่มีก็ได้ (\+\+)? เช่น harvex, harvex++ เป็นต้น
r"\bword\b"	แมตช์กับคำที่ขึ้นต้นด้วย "word" โดย \b หมายถึงไม่มีอักษระใดๆ เช่น "word" และไม่แมตช์กับ " word " หรือ "-word-" เป็นต้น

บทที่ 13

ไฟรอนกับฐานข้อมูล
(Python versus Database)

1. ฐานข้อมูลคืออะไร?

ฐานข้อมูล (Database) เป็นการเก็บรวบรวมข้อมูลที่มีความสัมพันธ์กันเข้าไว้ด้วยกันอย่างมีระบบ โดยข้อมูลที่ประกอบกันเป็นฐานข้อมูลนั้นจะต้องตรงตามวัตถุประสงค์การใช้งานขององค์กร ตัวอย่างของข้อมูลเช่น ชื่อ-นามสกุล ที่อยู่ หมายเลขโทรศัพท์ รายการสินค้า จำนวนสินค้าที่มีอยู่ในคลังสินค้า เป็นต้น

ระบบฐานข้อมูล (Database System) หมายถึงระบบที่รวบรวมข้อมูลต่างๆ ที่เกี่ยวข้องกันเข้าไว้ด้วยกันอย่างมีระบบและมีความสัมพันธ์ระหว่างข้อมูลต่างๆ ที่ชัดเจน ในระบบฐานข้อมูลจะประกอบด้วยแฟ้มข้อมูล โดยแต่ละแฟ้มข้อมูลจะถูกจัดเก็บเป็นระเบียน (Record) ซึ่งแต่ละระเบียนจะประกอบด้วยข้อมูลส่วนย่อยๆ หรือฟิลด์ (Field) หลายชิ้นที่มีความสัมพันธ์กันอย่างดังรูปที่ 13.1

Record		
Field	Field	Field
Student ID	Student Name	Student Address
5701345	Jason Yong	47/111 Tiwanon Road, Nonthaburi 11000

รูปที่ 13.1 แสดงระเบียนข้อมูล (Record) และฟิลด์ (Field)

จากรูปที่ 13.1 แสดงระเบียนข้อมูลที่ใช้จัดเก็บในแฟ้มข้อมูล ในตัวอย่างเป็นข้อมูลเกี่ยวกับนักศึกษาประกอบไปด้วย รหัสนักศึกษา (Student ID) เช่น 5701345, ชื่อ-นามสกุล (Student Name) เช่น "Jason Yong" และที่อยู่ (Student Address) เช่น "47/111 Tiwanon Road, Nonthaburi 11000" ตามลำดับ สำหรับรายละเอียดเกี่ยวกับการออกแบบระบบฐานข้อมูลและคำสั่ง SQL จะไม่อยู่ในขอบเขตของหนังสือเล่มนี้ ผู้อ่านสามารถอ่านเพิ่มเติมได้จากหนังสือเกี่ยวกับฐานข้อมูลที่มีจำหน่ายอยู่ทั่วๆ ไป

2. การใช้ Python กับฐานข้อมูล

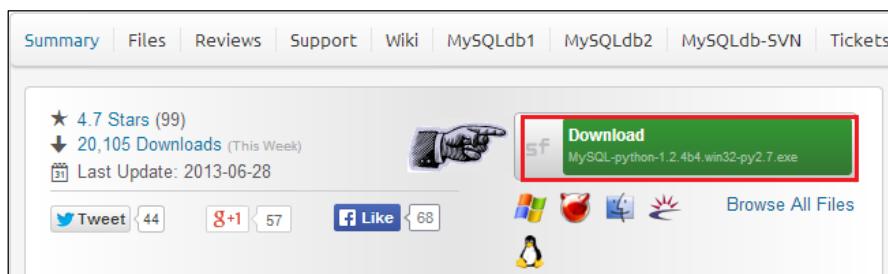
Python ได้จัดเตรียมเครื่องมือต่างๆ เพื่อสนับสนุนการจัดการเกี่ยวกับฐานข้อมูลไว้อย่างครบถ้วน เรียกว่า Python DB-API ซึ่งเป็นไลบรารีพื้นฐานสำหรับใช้ในการเขียนโปรแกรมร่วมกับฐานข้อมูล โดย Python รองรับการเชื่อมต่อ กับระบบฐานข้อมูล ได้มากหลายอย่าง อาทิเช่น GadFly, mSQL, MySQL, PostgreSQL, Microsoft SQL Server 2000, Informix, Interbase, Oracle และ Sybase เป็นต้น (สามารถดูรายละเอียดเพิ่มเติมได้ที่เว็บไซต์ <https://wiki.python.org/moin/DatabaseInterfaces>) แต่ในหนังสือเล่มนี้จะเน้นการเขียนโปรแกรมกับฐานข้อมูล MySQL เพราะเป็นฐานข้อมูลที่สามารถใช้งานได้ฟรีโดยไม่เสียค่าใช้จ่าย เป็นฐานข้อมูลที่ได้รับความนิยมมาก และมีประสิทธิภาพสูง ไม่แพ้ระบบฐานข้อมูลอื่นๆ

เมื่อต้องการเขียนโปรแกรม Python เชื่อมต่อกับฐานข้อมูลใดๆ ผู้เขียนโปรแกรมจำเป็นต้องดาวน์โหลดโมดูลสำหรับฐานข้อมูลนั้นๆ มาติดตั้งก่อน สำหรับในหนังสือนี้จะใช้ฐานข้อมูล MySQL ดังนั้น ผู้เขียนโปรแกรมสามารถดาวน์โหลดโมดูล MySQL ได้ที่ URL: <https://wiki.python.org/moin/MySQL> ดังรูปที่ 13.2



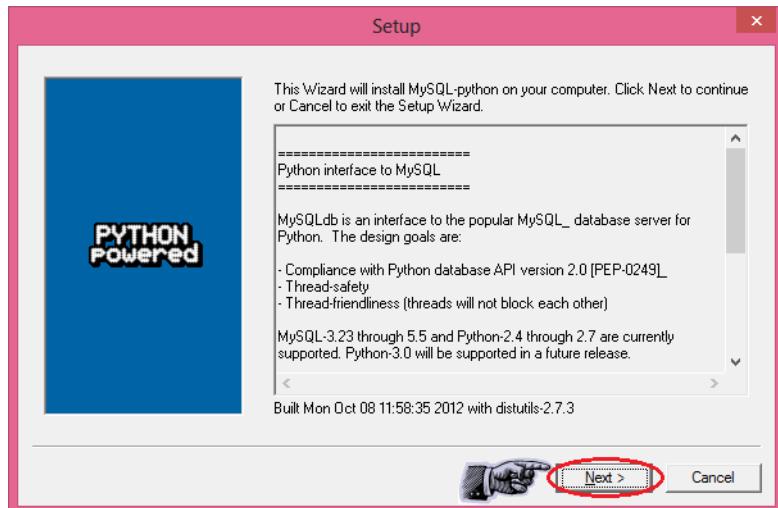
รูปที่ 13.2 ดาวน์โหลด Python โมดูลสำหรับฐานข้อมูล MySQL

ให้ผู้เขียนโปรแกรมคลิกดาวน์โหลดลิงค์ดังกล่าว จะปรากฏหน้าต่างเพื่อให้ดาวน์โหลดแฟ้มดังรูปที่ 13.3



รูปที่ 13.3 โมดูล MySQL-python-1.2.4b4.win32-py2.7.exe

ให้ผู้เขียนโปรแกรมคลิกเพื่อดาวน์โหลดแฟ้มชื่อ MySQL-python-1.2.xxx.win32-py2.7.exe (สำหรับวินโดวส์) เมื่อดาวน์โหลดแฟ้มดังกล่าวแล้ว ให้ทำการดับเบิลคลิกแฟ้มดังกล่าวเพื่อติดตั้งโมดูล MySQL ลงบนเครื่องคอมพิวเตอร์ (แนะนำให้ติดตั้งไฟร่อนเวอร์ชัน 2.7 เพื่อทำงานกับโมดูล MySQL เพราะโมดูล MySQL เวอร์ชันล่าสุดทำงานได้กับเวอร์ชัน 2.7 เท่านั้น)



รูปที่ 13.4 เริ่มต้นการติดตั้งโมดูล MySQL

ให้คลิกเลือก Next> ต่อไปเรื่อยๆ จนกว่าจะเสร็จสิ้นการติดตั้งโปรแกรม

ทดสอบการใช้งานโมดูล MySQL โดยทดสอบนำเข้า MySQLdb ดังตัวอย่างคำสั่งต่อไปนี้

```
>>> import MySQLdb
>>>
```

จากตัวอย่างด้านบนแสดงว่าโมดูล MySQLdb พร้อมใช้งานแล้ว แต่ถ้าโปรแกรมแสดงผลลัพธ์ดังต่อไปนี้แสดงว่ายังไม่มีการติดตั้ง MySQLdb

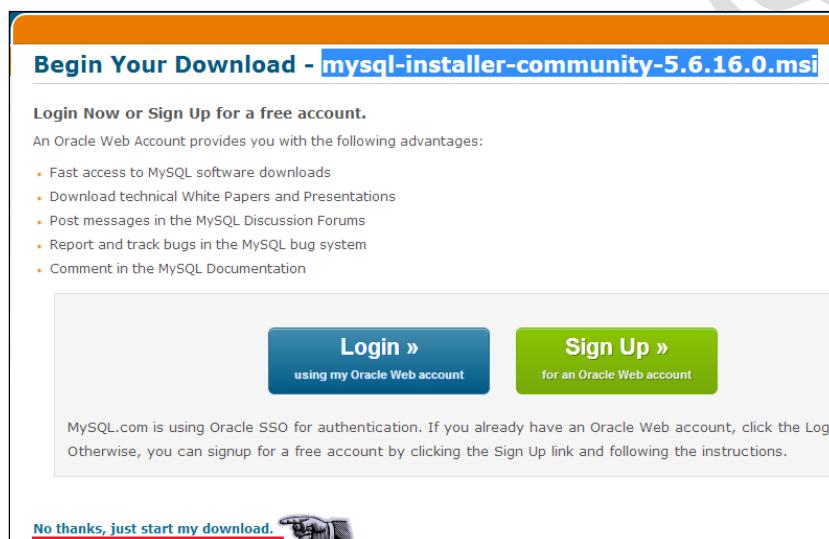
```
>>> import MySQLdb
Traceback (most recent call last):
File "test.py", line 3, in <module>
    import MySQLdb
ImportError: No module named MySQLdb
```

หลังจากติดตั้ง MySQLdb แล้ว จะเป็นต้องติดตั้งฐานข้อมูล MySQL ต่อไป โดยดาวน์โหลด MySQL ได้ที่ URL: <http://dev.mysql.com/downloads/windows/installer/> ดังรูปที่ 13.5



รูปที่ 13.5 ฐานข้อมูล MySQL Server

คลิก Download ดังรูปที่ 13.6



รูปที่ 13.6 ดาวน์โหลดฐานข้อมูล

เมื่อดาวน์โหลดฐานข้อมูล MySQL Server แล้ว ให้ทำการติดตั้งฐานข้อมูล โดยมีขั้นตอนดังนี้คือ

1. คลิกแฟ้มชื่อ mysql-installer-community-5.6.16.0.msi
2. เมนู License Agreement เลือก \Rightarrow I accept the license terms \Rightarrow Next >
3. เมนู Find lasted Products เลือก \Rightarrow Next >
4. เมนู Choosing a Setup Type เลือก \Rightarrow MySQL Server \Rightarrow Next >
5. เมนู Check Requirements เลือก \Rightarrow Next >
6. เมนู Installation Progress เลือก \Rightarrow Execute
7. เมนู Configuration Overview เลือก \Rightarrow Next >
8. เมนู MySQL Server Configuration เลือก \Rightarrow Server Configuration Type เลือก \Rightarrow Development Machine เลือก \Rightarrow Next >

9. เมนู MySQL Server Configuration \Rightarrow Root Account Password \Rightarrow ให้ผู้ใช้ใส่ Root Password และ Repeat Password, MySQL User Accounts \Rightarrow Add User ให้ผู้ใช้สร้าง User ตามที่ต้องการ \Rightarrow เลือก Next >
10. เมนู MySQL Server Configuration เลือก \Rightarrow Next > \Rightarrow คลิก Next ไปเรื่อยๆ จนกว่าจะ finish

ทดสอบการเชื่อมต่อกับฐานข้อมูล MySQL โดยเขียนโปรแกรมภาษาไพธอนดังตัวอย่างโปรแกรมที่ 13.1

Program Example 13.1: testing database connection

```

⇒1 import MySQLdb
2   # Open database connection
⇒3 db = MySQLdb.connect("localhost","root","abc123","test" )
⇒4   # prepare a cursor object using cursor() method
cursor = db.cursor()
5   # execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")
6   # Fetch a single row using fetchone() method.
data = cursor.fetchone()
7   print ("Database version : %s" % data)
8   # disconnect from server
9 db.close()
10
⇒11

```



Database version : 5.6.16

จากตัวอย่างโปรแกรมที่ 13.1 ทดสอบการเชื่อมต่อกับฐานข้อมูล MySQL โดยเริ่มต้นในบรรทัดที่ 1 โปรแกรมนำเข้าโมดูล MySQLdb เพื่อใช้สำหรับจัดการกับฐานข้อมูล บรรทัดที่ 2 โปรแกรมทำการเชื่อมต่อฐานข้อมูลโดยใช้เมธอด connect โดยเมธอดดังกล่าวต้องการพารามิเตอร์ 4 ตัวคือ ที่อยู่ของฐานข้อมูล (หมายเลขไอพี ในตัวอย่างคือ localhost หรือ 127.0.0.1), ชื่อผู้ใช้งาน (User Name) มีค่าเท่ากับ root, รหัสผ่าน (Password) มีค่าเท่ากับ "abc123" (เป็นรหัสผ่านในขั้นตอนการติดตั้ง) และชื่อฐานข้อมูล สำหรับตัวอย่างคือ "test" ผลลัพธ์ที่ได้จากการเชื่อมต่อจะเก็บไว้ในตัวแปร db บรรทัดที่ 4 โปรแกรมกำหนดให้สามารถสั่งงานฐานข้อมูลโดยผ่านคำสั่ง SQL ได้ โดยใช้เมธอด cursor() บรรทัดที่ 6 ทดสอบโดยการสอบถามเวอร์ชันของฐานข้อมูล ด้วยคำสั่ง SQL คือ "SELECT VERSION()" ด้วยเมธอด execute() บรรทัดที่ 8 โปรแกรมสั่งประมวลผลคำสั่ง SQL ให้ดึงข้อมูลจากฐานข้อมูลด้วยเมธอด fetchone() แต่จะดึงข้อมูลมาแสดงผลครั้งละ 1 ระเบียบ

ผลลัพธ์ที่เกิดจากการประมวลผลคือ โปรแกรมสั่งพิมพ์เวอร์ชันของฐานข้อมูล (บรรทัดที่ 9) มีค่าเท่ากับ "Database version : 5.6.16" บรรทัดที่ 11 โปรแกรมปิดฐานข้อมูลด้วยเมธอด close()

3. การสร้างฐานข้อมูลและตาราง (Creating Database & Table)

ลำดับการดำเนินงานกับฐานข้อมูลจะมีอยู่ 4 ขั้นตอนดังนี้คือ

1. ประกาศหรือนำเข้าโมดูลเพื่อดำเนินการกับฐานข้อมูล (โมดูล MySQL) คือ
import MySQLdb
2. เชื่อมต่อกับฐานข้อมูล ซึ่งมีรูปแบบคำสั่งคือ

```
MySQLdb.connect(database_address, user_name, password)
```

หรือ

```
MySQLdb.connect(database_address, user_name, password, database_name)
```

โดย database_address คือ ที่อยู่ของฐานข้อมูล (IP Address), user_name คือ ชื่อผู้ใช้งาน, password คือ รหัสผ่าน และ database_name คือ ชื่อฐานข้อมูลที่ต้องการใช้งาน

3. ออกคำสั่งเพื่อใช้งานฐานข้อมูลด้วยคำสั่ง SQL หรือ Store Procedures (การเขียน SQL ฝังไว้ที่ฐานข้อมูลเพื่อให้ช่วยแบ่งเบาภาระของเว็บเซิฟเวอร์) โดยใช้เมธอด execute() เช่น

```
sql = "SHOW DATABASES"
execute(sql)
# execute("SHOW TABLES")
```

4. ปิดการเชื่อมต่อฐานข้อมูล ด้วยเมธอด close() เช่น

```
db.close()
```

เมื่อผู้เขียนโปรแกรมสามารถเชื่อมต่อกับฐานข้อมูลเรียบร้อยแล้ว สามารถสร้างฐานข้อมูลตาราง หรือเปลี่ยนด้วยเมธอด execute() ได้ทันที ดังโปรแกรมตัวอย่างที่ 13.2

Program Example 13.2: Creating database and use it

⇒1	import MySQLdb
2	# Connect database
⇒3	db = MySQLdb.connect ("localhost", "root", "abc123")
⇒4	cursor = db.cursor()
5	
6	# Create and use database
⇒7	cursor.execute ("CREATE DATABASE DBTest")
⇒8	cursor.execute ("USE DBTest")
9	
10	# Create table Employee
⇒11	sql = """CREATE TABLE EMPLOYEE (
	FIRST_NAME CHAR(20) NOT NULL,
	LAST_NAME CHAR(20),
	AGE INT,
	SEX CHAR(1),

```

    INCOME FLOAT) """
12
⇒13 cursor.execute(sql)
14
15 # Shows table Employee
⇒16 cursor.execute("SHOW TABLES")
⇒17 data = cursor.fetchall()
⇒18 print(data)
19
20 # disconnect from server
⇒21 db.close()

```



OUTPUT

```

mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| dbtest        |
| mysql          |
| performance_schema |
| test           |
+-----+
5 rows in set (0.00 sec)

```

จากตัวอย่างโปรแกรมที่ 13.2 แสดงการสร้างตารางและแสดงผลจากการสร้างตารางข้อมูล

เริ่มต้นบรรทัดที่ 1 โปรแกรมนำเข้าโมดูล MySQLdb เพื่อใช้ประมวลผลกับฐานข้อมูล บรรทัดที่ 3

โปรแกรมเชื่อมต่อระบบฐานข้อมูลผ่านเมธอด connect() ซึ่งเมธอดดังกล่าวต้องการพารามิเตอร์ 3 ตัว คือ ที่อยู่ฐานข้อมูล (localhost หรือ 127.0.0.1) พารามิเตอร์ตัวที่ 2 คือ ชื่อผู้ใช้ (root) และพารามิเตอร์ตัวที่ 3 คือ รหัสผ่าน (abc123) เมื่อสามารถเชื่อมต่อฐานข้อมูลได้สำเร็จ โปรแกรมจะใช้เมธอด cursor() เพื่อให้ผู้ใช้งานสามารถประมวลผลผ่านคำสั่ง SQL ได้ (บรรทัดที่ 4) เมื่อฐานข้อมูลให้สิทธิกับผู้ใช้ในการ ส่งคำสั่ง SQL เพื่อประมวลผลได้แล้ว ลำดับต่อไปโปรแกรมจะทำการสร้างฐานข้อมูลชื่อ "DBTest" (บรรทัดที่ 7) ถ้าการสร้างฐานข้อมูลไม่เกิดข้อผิดพลาดใดๆ โปรแกรมจะสั่งให้ฐานข้อมูลดังกล่าวที่ สร้างขึ้นโดยใช้คำสั่ง "USE DBTest" (บรรทัดที่ 8)

บรรทัดที่ 11 โปรแกรมสร้างคำสั่ง sql เพื่อสร้างตารางข้อมูลชื่อ "EMPLOYEE" โดยมีพิวเด็ต ข้อมูลต่างๆ ดังนี้ คือ FIRST_NAME, LAST_NAME, AGE INT, SEX CHAR และ INCOME FLOAT และประมวลผลคำสั่งดังกล่าวในบรรทัดที่ 13, ลำดับถัดไปบรรทัดที่ 16 โปรแกรมใช้คำสั่ง "SHOW TABLES" พร้อมกระตุ้นให้ฐานข้อมูลดึงข้อมูลจากฐานข้อมูลมาแสดงผลด้วยเมธอด fetchall() โดยดึง ข้อมูลทุกรายบุคคลมาแสดงผล (บรรทัดที่ 17)

สำหรับผลลัพธ์ของการแสดงผลแสดงดัง OUTPUT ด้านบน ในบรรทัดสุดท้ายโปรแกรมสั่งปิด ฐานข้อมูลด้วยเมธอด close()



Note: ในการเชื่อมต่อกับระบบฐานข้อมูลจำเป็นต้องระบุที่อยู่ของฐานข้อมูล ซึ่งเรียกว่า IP Address (Internet Protocol Address) ถ้าผู้เขียนโปรแกรมเชื่อมต่อฐานข้อมูลที่ติดตั้งอยู่ในเครื่องของผู้เขียนโปรแกรมเอง จะสามารถใช้คำว่า localhost ซึ่งจะมีหมายเลข IP Address คือ 127.0.0.1 แต่ถ้าผู้เขียนโปรแกรมทำการเชื่อมต่อฐานข้อมูลในระบบใกล้ จะต้องเปลี่ยน IP Address จาก 127.0.0.1 เป็นหมายเลข IP ที่ต้องการ

4. คำสั่งดำเนินการเกี่ยวกับฐานข้อมูล

การดำเนินการพื้นฐานหลัก ๆ เกี่ยวกับฐานข้อมูลมี 6 คำสั่งคือ การเพิ่มระเบียน (Insert), การลบระเบียน (Delete), การปรับปรุงระเบียน (Update), การอ่านระเบียน (Read), การยืนยันคำสั่ง (Commit) และ การยกเลิกการทำงานของคำสั่งก่อนหน้า (Rollback) ซึ่งมีรายละเอียดแต่ละคำสั่งดังต่อไปนี้

1. คำสั่งการเพิ่มระเบียน (Insert Operation)

โปรแกรมตัวอย่างที่ 13.3 แสดงการใช้งานคำสั่ง Insert ซึ่งมีรายละเอียดดังนี้

Program Example 13.3: Insert Operation

```

1 import MySQLdb
2
3 db = MySQLdb.connect("127.0.0.1","root","abc123","DBTest" )
4 cursor = db.cursor()
5 sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
6     LAST_NAME, AGE, SEX, INCOME)
7     VALUES ('John', 'Carter', 20, 'M', 2000)"""
8
9 try:
10     cursor.execute(sql)
11     # Commit your changes in the database
12     db.commit()
13 except:
14     # Rollback in case there is any error
15     db.rollback()
16 db.close()
```

ทดสอบผลลัพธ์โดยสั่งงานผ่าน MySQL Command line client ดังนี้คือ



```

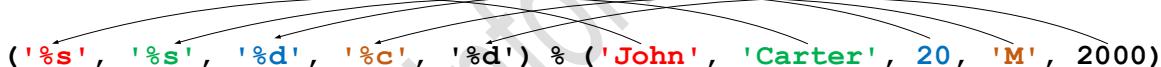
mysql> use DBTest;
Database changed
mysql> show tables;
+-----+
| Tables_in_dbtest |
+-----+
| employee         |
+-----+
1 row in set (0.00 sec)
mysql> select * from employee;
+-----+-----+-----+-----+-----+
| FIRST_NAME | LAST_NAME | AGE   | SEX   | INCOME |
+-----+-----+-----+-----+-----+
```

John	Carter	20	M	2000
1 row in set (0.00 sec)				

จากตัวอย่างโปรแกรมที่ 13.3 แสดงการใช้คำสั่ง `insert` เพื่อเพิ่ม纪录 (Record) ลงฐานข้อมูล เริ่มต้นในบรรทัดที่ 3 โปรแกรมเชื่อมต่อฐานข้อมูลชื่อว่า DBTest ด้วยชื่อผู้ใช้คือ "root" และรหัสผ่านคือ "abc123" สำหรับคำสั่งที่ต้องการประมวลผลคือ ต้องการเพิ่ม纪录ใหม่เข้าไปยังฐานข้อมูลชื่อ DBTest ในตารางชื่อ Employee (บรรทัดที่ 5 และ 8) โดยมีพิวาร์ต่างๆ ประกอบไปด้วย `FIRST_NAME = "John", LAST_NAME = "Carter", AGE = "20", SEX = "M" และ INCOME = "2000"` บรรทัดที่ 6 โปรแกรมใช้คำสั่ง `try...except` เพื่อดักจับความผิดพลาดที่อาจจะเกิดจากการประมวลผลคำสั่ง SQL ถ้า การประมวลผลจากคำสั่ง SQL ทำงานเป็นปกติโดยไม่มีความผิดพลาด โปรแกรมจะทำการยืนยันคำสั่ง ที่ทำว่าถูกต้องแล้วด้วยเมธอด `commit` (บรรทัดที่ 9) แต่ถ้ามีความผิดพลาดเกิดขึ้น โปรแกรมจะทำงาน ในบรรทัดที่ 12 คือการยกเลิกคำสั่งที่ได้กระทำไปก่อนหน้าคืนกลับสู่ภาวะเดิม

จากตัวอย่างในบรรทัดที่ 5 คือคำสั่งเพิ่ม纪录ของ SQL สามารถเขียนใหม่ได้อีกแบบดังนี้คือ

```
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
    LAST_NAME, AGE, SEX, INCOME) \
VALUES ('%s', '%s', '%d', '%c', '%d') % \
    ('John', 'Carter', 20, 'M', 2000)
```



2. คำสั่งการอ่าน纪录 (Read Operation)

โดยปกติการอ่านหรือดึงข้อมูลจากฐานข้อมูลมาแสดงผล จะมีเมธอดที่สำคัญคือ `fetchone()` และ `fetchall()` โดย `fetchone()` จะดึงข้อมูลที่เกิดจากคำสั่ง SQL มาแสดงผลครั้งละ 1 ระเบียบ ส่วนเมธอด `fetchall()` จะดึงข้อมูลที่เกิดจากคำสั่ง SQL มาแสดงผลทุกระเบียบ พร้อมยังได้จัดเตรียมแอตทริบิวต์ที่ ทำหน้าที่สำหรับนับจำนวน纪录ที่เกิดจากคำสั่ง SQL ไว้ด้วยคือ `rowcount` ซึ่งแอตทริบิวต์ดังกล่าวจะ ทำงานเมื่อเรียกใช้งานเมธอด `execute()` สำหรับโปรแกรมตัวอย่างที่ 13.4 จะแสดงการอ่านค่าจาก ฐานข้อมูล ซึ่งมีรายละเอียดดังนี้

Program Example 13.4: Read Operation (`fetch`, `fetchall`)

```

1 import MySQLdb
2 db = MySQLdb.connect("127.0.0.1", "root", "abc123", "DBTest" )
3 cursor = db.cursor()
⇒4 sql = "SELECT * FROM EMPLOYEE \
        WHERE INCOME > '%d'" % (1000)
5 try:
6     cursor.execute(sql)
7     results = cursor.fetchall()

```

```

⇒8   for row in results:
⇒9       fname = row[0]
⇒10      lname = row[1]
⇒11      age = row[2]
⇒12      sex = row[3]
⇒13      income = row[4]
⇒14      print
("fname=%s, lname=%s, age=%d, sex=%s, income=%d, rowcount=%d") % \
                (fname, lname, age, sex, income, cursor.rowcount)
15 except:
⇒16     print ("Error: unable to fetch data")
17 db.close()

```



OUTPUT fname=John, lname=Carter, age=20, sex=M, income=2000, rowCount=1

จากตัวอย่างโปรแกรมที่ 13.4 แสดงการใช้เมธอด `fetchall()` เพื่อดึงข้อมูลจากฐานข้อมูลมาแสดงผล ในบรรทัดที่ 4 เป็นคำสั่ง SQL ที่ร้องขอให้ฐานข้อมูล DBTest แสดงข้อมูลที่อยู่ในตาราง `EMPLOYEE` ที่มีค่าในพิลเดอร์ `INCOME` มากกว่า 1000 ออกมาระบุผล ผลลัพธ์ที่ได้จากการประมวลผลคำสั่ง SQL ดังกล่าวจะถูกอ่านด้วยเมธอด `fetchall()` ในบรรทัดที่ 7 ผลลัพธ์ที่ได้จากการเมธอด `fetchall()` อาจจะมีมากกว่า 1 จะเปลี่ยนเก็บไว้ในตัวแปร `results` ดังนั้นจำเป็นต้องใช้คำสั่ง `for` (บรรทัดที่ 8) ดึงข้อมูลเหล่านี้มาแสดงผล โดย `row[0]` เก็บข้อมูลชื่อคือ "John", `row[1]` เท่ากับ "Carter", `row[2]` หรือ `age` เท่ากับ "20", `row[3]` หรือ `sex` เท่ากับ "M" และ `row[4]` หรือ `income` เท่ากับ 2000 ดังในบรรทัดที่ 9 ถึง 14 แต่ถ้าการประมวลผลคำสั่ง SQL เกิดข้อผิดพลาดโปรแกรมพิมพ์ข้อความ "Error: unable to fetch data" ออกจอภาพ (บรรทัดที่ 16)

3. คำสั่งการปรับปรุงระเบียน (Update Operation)

จะเปลี่ยนที่อยู่ในฐานข้อมูลสามารถปรับปรุงได้โดยการใช้เมธอด `execute()` เช่นเดียวกับการดำเนินการอื่นๆ ดังตัวอย่างที่ 13.5

Program Example 13.5: Update Operation

```

1 import MySQLdb
2 db = MySQLdb.connect("127.0.0.1", "root", "abc123", "DBTest" )
3 cursor = db.cursor()
4 # Prepare SQL query to UPDATE required records
5 sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" %
('M')
6 try:
7     cursor.execute(sql)
8     db.commit()
9 except:
10    db.rollback()
11 db.close()

```

แสดงผลลัพธ์ด้วยคำสั่ง `select * from employee;` ใน MySQL Command Line Client ดังนี้



OUTPUT

```
mysql> select * from employee;
+-----+-----+-----+-----+
| FIRST_NAME | LAST_NAME | AGE | SEX | INCOME |
+-----+-----+-----+-----+
| John       | Carter    | 21  | M   | 2000  |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

จากตัวอย่างโปรแกรมที่ 13.5 แสดงการปรับปรุงฐานข้อมูลที่เป็นเพศชาย "M" ให้มีอายุเพิ่มขึ้น อีก 1 ปี (SET AGE = AGE + 1 ในบรรทัดที่ 5) ด้วยคำสั่ง SQL คือ "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M') โดย %c จะถูกแทนที่ด้วยอักษร 'M' สำหรับการทำงานของโปรแกรมในบรรทัดอื่นๆ จะเหมือนกับตัวอย่างที่ 13.4 สำหรับในตัวอย่างนี้โปรแกรมจะไม่มีการแสดงผลลัพธ์ใดๆ ออกทางจอภาพ ดังนั้นผู้เขียนโปรแกรมสามารถแสดงผลการทำงานได้โดยผ่านทาง MySQL Command Line Client ด้วยคำสั่ง `select * from employee;` ผลลัพธ์แสดงในตัวอย่าง OUTPUT ด้านบน

4. คำสั่งการลบระเบี่ยน (Delete Operation)

คำสั่งลบระเบี่ยนข้อมูลจะใช้เมื่อผู้เขียนโปรแกรมต้องการลบระเบี่ยนออกจากฐานข้อมูล โดยใช้งานผ่านเมธอด `execute()` เช่นเดียวกัน ดังตัวอย่างที่ 13.6

Program Example 13.6: Delete Operation

```
1 import MySQLdb
2 db = MySQLdb.connect("127.0.0.1","root","abc123","DBTest" )
3 cursor = db.cursor()
4 # Prepare SQL query to DELETE required records
5 ⇒sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
6 try:
7     cursor.execute(sql)
8     db.commit()
9 except:
10    db.rollback()
11 db.close()
```

แสดงผลลัพธ์ด้วยคำสั่ง `select * from employee;` ใน MySQL Command Line Client ดังนี้



OUTPUT

```
mysql> select * from employee;
Empty set (0.00 sec)
```

จากตัวอย่างโปรแกรมที่ 13.6 แสดงการลบระเบี่ยนข้อมูลของผู้ที่มีอายุมากกว่า 20 ปีขึ้นไป ออกจากฐานข้อมูล ด้วยคำสั่ง SQL เท่ากับ "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20) โดยสัญลักษณ์ "%d" % (20) หมายถึง %d จะถูกแทนที่ด้วย 20 จากซึ่งเกิดจาก (20) ดังรูปด้านล่าง

"DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)

5. การยืนยันและยกเลิกคำสั่งการดำเนินการ (Commit and Rollback Operation)

การดำเนินการกับระบบฐานข้อมูลมีชื่อเรียกอีกอย่างคือ Transaction หมายถึง กลุ่มคำสั่งที่ดำเนินการต่อข้อมูลบนฐานข้อมูล ซึ่งจะถูกมองเป็นหน่วยการทำงานเพียงหนึ่งหน่วยซึ่งไม่สามารถที่จะแบ่งแยกย่อยได้ เพื่อเป็นการยืนยันความถูกต้องของข้อมูล ประกอบไปด้วย 4 ข้อคือ

- 1) Atomicity (แบ่งแยกไม่ได้) Transaction จะต้องเสร็จสิ้นอย่างสมบูรณ์ หรือไม่ เช่นนั้นก็จะไม่เกิดขึ้นเลย เช่น A ต้องการโอนเงินให้ B เป็นจำนวนเงิน 500 บาท คำสั่งที่ต้องการในการดำเนินการคือ ถอนเงินจากบัญชีของ A 500 บาทแล้วฝากเงินเข้าไปในบัญชีของ B เท่ากับ 500 บาท แต่ถ้ามีการดำเนินการเพียงหนึ่งคำสั่งคือถอนเงินออกจากบัญชีของ A แต่ไม่มีการฝากเงินเข้าไปในบัญชีของ B ก็จะทำให้ฐานข้อมูลอยู่ในสภาพที่ไม่ถูกต้อง เนื่องจากเงินในบัญชีของ A ได้หายไปแล้ว 500 บาท แต่เงินในบัญชีของ B ไม่ได้เพิ่มขึ้น 500 บาท อย่างที่ควรจะเป็น ดังนั้นถ้าหากมีการดำเนินการเพียงหนึ่งคำสั่งและระบบเกิดหยุดชะงักไม่ว่าด้วยสาเหตุใดๆ ก็ตาม Transaction นั้นจะต้องถูกทำให้ย้อนกลับไป (roll back) ณ จุดที่ไม่เคยมีการกระทำใดๆ ตาม Transaction นั้นเกิดขึ้นมาก่อน
- 2) Consistency (ความสอดคล้องถูกต้อง) หมายถึงฐานข้อมูลจะต้องคงอยู่ในสถานะที่ถูกต้องเสมอ ไม่ว่าก่อนที่จะเกิด Transaction หรือหลังจากที่มีการดำเนินการตาม Transaction และ การดำเนินการของ Transaction จะต้องไม่มีผลทำให้ฐานข้อมูลสูญเสียความถูกต้องและความสอดคล้องของข้อมูล
- 3) Isolation (การไม่ถูกรบกวน) โดยที่ไปในระบบฐานข้อมูลนั้น มักจะมี Transaction ที่ดำเนินการพร้อมๆ กัน จำนวนหลาย Transaction โดยที่ Transaction เหล่านั้นในบางครั้งก็ใช้ข้อมูลชุดเดียวกัน ซึ่งการใช้ข้อมูลร่วมกันของ Transaction เหล่านั้น ก็อาจจะทำให้การทำงานของ Transaction ได้ผลลัพธ์ที่ไม่ถูกต้องได้ ดังนั้นในการทำงานของแต่ละ Transaction จะต้องถูกจัดลำดับการทำงานให้ไม่มีการรบกวนกันโดยให้แต่ละ Transaction สามารถทำงานโดยเชื่อมว่ามีเพียง Transaction เดียวเท่านั้น ที่กำลังทำงานอยู่
- 4) Durability (ความคงทน) เมื่อ Transaction ได้ดำเนินการเสร็จสมบูรณ์แล้ว ผลลัพธ์ที่เกิดจาก Transaction นั้นบนฐานข้อมูล จะต้องคงอยู่ตลอดไป ไม่ว่าจะเกิดปัญหาระบบทหยุดชะงัก (System crash) หรือระบบล้มเหลวในการทำงาน (System

failure) ข้อมูลที่เป็นผลลัพธ์ที่เกิดจาก Transaction ที่ทำงานอย่างถูกต้องนั้น จะต้องคงอยู่ตลอดไปเสมอ หลังจากที่ระบบได้รับการรักษาให้กลับมาทำงาน (Recovery) อีกครั้ง

จากที่ได้กล่าวมาแล้วเบื้องต้น ฐานข้อมูลจำเป็นต้องมีการยืนยันและยกเลิก Transaction ซึ่งไฟรอ念เตรียมคำสั่งไว้เพื่อให้ผู้เขียนโปรแกรมเรียกใช้งานได้คือ Commit และ Rollback ซึ่งมีรูปแบบคำสั่งคือ

คำสั่งยืนยันการดำเนินการ (Commit)

`db.commit()`

คำสั่งยกเลิกการดำเนินการ (Rollback)

`db.rollback()`

สำหรับตัวอย่างการใช้งานทั้ง 2 คำสั่ง ผู้เขียนได้แสดงไว้แล้วในตัวอย่างต่างๆ ที่ผ่านมา โดยมีหลักการใช้งานคือ ถ้าการประมวลผลคำสั่ง SQL ไม่มีความผิดพลาดใดๆ เกิดขึ้นในขณะทำ Transaction ให้เรียกใช้เมธอด commit() เพื่อยืนยันว่าคำสั่งที่สั่งงานไปแล้วนั้นสมบูรณ์ แต่ถ้ามีความผิดพลาดเกิดขึ้นในขณะทำ Transaction ให้เรียกใช้เมธอด rollback() เพื่อยกเลิกคำสั่งที่ทำงานไปแล้วกลับคืนมา โดยต้องทำงานร่วมกับคำสั่ง try...except เสมอ (ถ้าผู้อ่านไม่มีความรู้พื้นฐานเกี่ยวกับคำสั่งดังกล่าว สามารถอ่านเพิ่มเติมได้ในบทที่ 9)

6. คำสั่งยกเลิกการเชื่อมต่อฐานข้อมูล (Disconnect)

หลังจากที่มีการใช้งานแฟ้มได้ ก็ตาม ขั้นตอนสุดท้ายคือการปิดแฟ้ม เพื่อบอกให้ระบบปฏิบัติการทราบว่าไม่ต้องการใช้งานแฟ้มดังกล่าวแล้ว ระบบปฏิบัติการก็จะคืนหน่วยความจำที่ใช้จัดเก็บแฟ้มดังกล่าวคืนกลับสู่ระบบต่อไป ในการใช้งานฐานข้อมูลก็มีลักษณะเดียวกันคือ จำเป็นต้องยุติการเชื่อมต่อฐานข้อมูลเมื่อไม่มีการใช้งานได้ กับฐานข้อมูลแล้ว โดยใช้เมธอด close() ซึ่งมีรูปแบบคำสั่งดังนี้

`db.close()`

สำหรับตัวอย่างการใช้งานเมธอด close() ผู้เขียนได้แสดงไว้แล้ว จากตัวอย่าง 13.1 – 13.6 ที่อยู่ด้านบน

บทที่ 14

เว็บและซีจีไอ

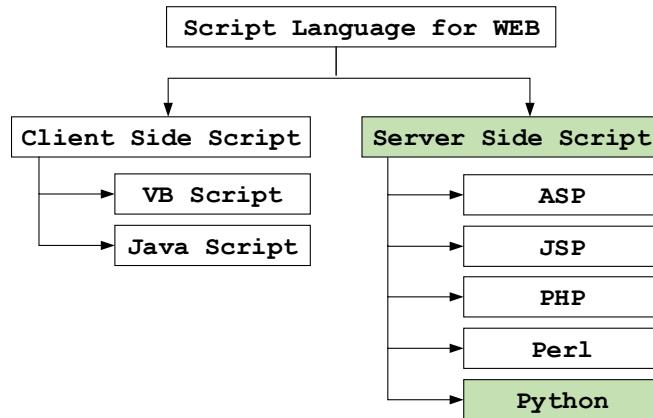
(Web and Common Gateway Interface:CGI)



1. เว็บและซีจีไอคืออะไร?

เว็บ (Web) ย่อมาจากเวลต์ไวด์เว็บ (World Wide Web) หมายถึงระบบคอมพิวเตอร์ที่ให้บริการข้อมูล (เอกสาร HTML หรือเว็บเพจ) ซึ่งข้อมูลดังกล่าวสามารถใช้งานผ่านทางเครือข่ายอินเทอร์เน็ต (Internet) ได้ โดยใช้โปรโทคอลเช่นทีทีพี (HTTP หรือ Hypertext Transfer Protocol) ในการสื่อสาร ซึ่งเอกสาร HTML เปรียบเสมือนกับสมุดเล่มหนึ่ง ที่ไม่มีการตอบโต้กับผู้ใช้งาน ทำให้เว็บเพจดูไม่น่าสนใจ แต่ถ้าผู้เขียนโปรแกรมต้องการให้เว็บเพจดังกล่าวมีความน่าสนใจเพิ่มขึ้น เช่น แสดงตัวนับ (Counter) ที่จะทำให้รู้ว่ามีคนมาเยี่ยมชมโดยเพียงแค่กดหนึ่ง หรือมีเว็บบอร์ด (Web Board) สำหรับให้ผู้เข้ามาเยี่ยมชมสามารถเขียนคำความทึ่งเอาไว้ก็ได้ เป็นต้น สิ่งต่างๆ ที่กล่าวมาเหล่านี้ HTML ธรรมดามาไม่สามารถทำได้ จำเป็นต้องอาศัยภาษาสคริปต์ (Scripting Language) เพื่อช่วยให้เว็บเพจมีความสามารถมากขึ้น ในการเขียนภาษาสคริปต์ สำหรับเว็บไซต์นั้นเขียนได้หลายภาษา และมีรูปแบบการเขียนอยู่ด้วยกันสองแบบ คือ

1. Client-Side Scripting เป็นการเขียนโปรแกรมภาษาสคริปต์ ให้ทำงานบนเว็บบราวเซอร์ (Web Browser) โดยเขียนโปรแกรมแทรกหรือฝัง (Embed) เข้าไปเป็นส่วนหนึ่งของเอกสาร HTML โปรแกรมภาษาสคริปต์ประเภทนี้ได้แก่ JavaScript, VBScript
2. Server-Side Scripting เป็นการเขียนโปรแกรมภาษาสคริปต์ ให้ทำงานบนเว็บเซิร์ฟเวอร์ (Web Server) โดยเว็บบราวเซอร์จะเป็นเพียงแค่ตัวที่แสดงผลการทำงานเท่านั้น โปรแกรมที่ทำงานบนเว็บเซิร์ฟเวอร์เหล่านี้เรียกว่า CGI Script ซึ่งสามารถเขียนได้หลายภาษาด้วยกัน เช่น Perl, JSP, ASP, PHP และ Python

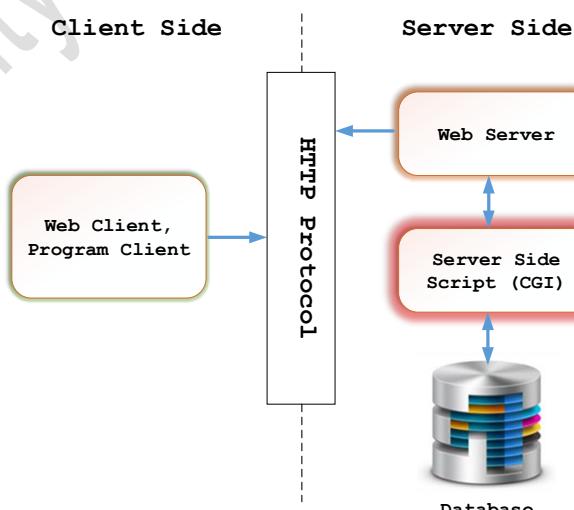


รูปที่ 14.1 แสดงประเภทของภาษาสคริปต์ที่ทำงานกับเว็บ

CGI ย่อมาจาก Common Gateway Interface หมายถึงวิธีการติดต่อสื่อสารที่ใช้ระหว่างเว็บเซิร์ฟเวอร์และโปรแกรมผู้ไคลเอนต์ (Client program) ซึ่งไม่จำกัดภาษาที่ใช้เขียน และไม่ขึ้นต่อระบบปฏิบัติการใดๆ ข้อสำคัญคือ โปรแกรมผู้ไคลเอนต์เหล่านั้นจะต้องรับและส่งข้อมูลตามรูปแบบที่ CGI กำหนดไว้ เราจึงเรียกโปรแกรมที่ทำงานบนเว็บเซิร์ฟเวอร์โดยใช้มาตรฐานการสื่อสารแบบ CGI ว่า CGI Script นั้นเอง

สถาปัตยกรรมของ CGI

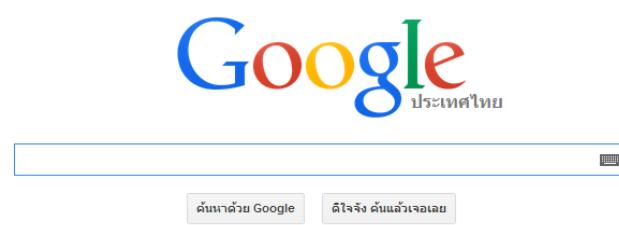
รูปที่ 14.2 แสดงสถาปัตยกรรมการทำงานของ CGI ซึ่งประกอบไปด้วยโปรแกรมทางผู้ไคลเอนต์ (Client Side) คือ เว็บไคล์เอนต์ (Web Client) หรือ โปรแกรมไคล์เอนต์ (Client Program) และโปรแกรมผู้เซิร์ฟเวอร์ (Server Side) ที่ทำงานร่วมกันระหว่างเว็บเซิร์ฟเวอร์ (Web Server), ซีจีไอ (CGI) และฐานข้อมูล (Database) โปรแกรมทั้งสองฝ่ายจะอาศัยโปรโตคอล HTTP ในการติดต่อสื่อสารระหว่างกัน



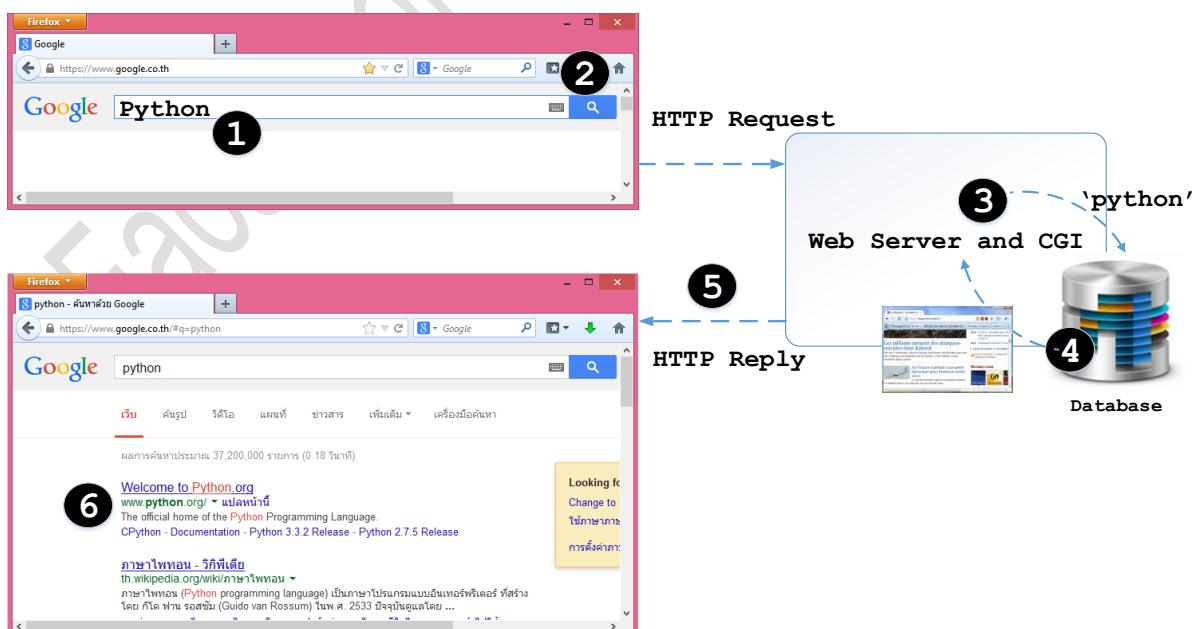
รูปที่ 14.2 แสดงสถาปัตยกรรมของ CGI

การทำงานของ CGI

CGI มีหน้าที่ประมวลผลข้อมูลที่ร้องขอมาจากโปรแกรมฝั่งไคลเอนต์ เช่น เว็บไคลเอนต์หรือโปรแกรมไคลเอนต์ โดยแสดงผลออกมารูปแบบ HTML ร่วมกับสคริปต์ เรียกว่าโฮมเพจ (Home page) ยกตัวอย่างเช่น เว็บไซต์ Google (www.google.com) เป็นเว็บไซต์ที่ให้บริการค้นหาข้อมูลที่มีผู้ใช้บริการมากที่สุดในโลก เว็บไซต์ดังกล่าวจะเตรียมช่องสำหรับรับข้อความที่ผู้ใช้สามารถพิมพ์ข้อความที่ต้องการค้นหาอะไรมาก็ได้ลงไป จากนั้นกดปุ่มค้นหา (Search) สักครู่ Google จะแสดงผลข้อมูลที่ต้องการค้นหาอออกมาให้



เว็บไซต์ Google ทำงานอย่างไร? คำตอบคือ บนเว็บไซต์ Google นั้นจะมี CGI ฝังตัวอยู่กับเว็บเซิร์ฟเวอร์ เมื่อผู้ใช้งานเปิดโปรแกรมเว็บไคลเอนต์ (IE, Firefox, Chrome) ไปยังเว็บไซต์ของ Google และพิมพ์ข้อความที่ต้องการค้นหา เช่น “Python” เมื่อกดปุ่มค้นหา (หรือเรียกว่าการ Submit ฟอร์ม) ข้อมูลทั้งหมดจะถูกส่งไปให้กับ CGI บนเซิร์ฟเวอร์ทำงาน CGI จะนำคำว่า “Python” ไปค้นหาในฐานข้อมูลว่ามีข้อความดังกล่าวหรือไม่ ถ้า CGI พบรูปแบบดังกล่าว จะส่งผลลัพธ์ผ่านพอร์ต HTTP กลับไปยังเว็บไคลเอนต์ของผู้ใช้งาน ดังรูปที่ 14.3



รูปที่ 14.3 แสดงการทำงานของ CGI

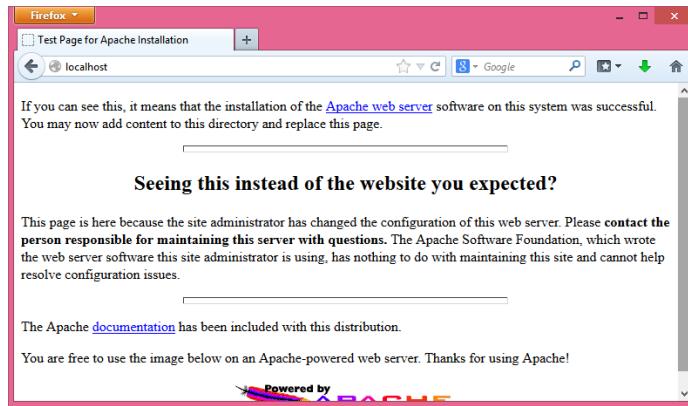
จากรูปที่ 14.3 ❶ เปิดเว็บไคลเอนต์ (Firefox, Crome, IE) และพิมพ์ www.google.com ในช่อง address จะปรากฏหน้าต่างของเว็บไซต์ google ในบราวเซอร์ ให้ผู้ใช้พิมพ์ข้อความที่ต้องการค้นหา จากตัวอย่างคือคำว่า “python” \Rightarrow ❷ กดปุ่มค้นหาเพื่อร้องขอไปยัง CGI ที่ผ่านเว็บเซิร์ฟเวอร์ด้วยโปรโตคอล HTTP (Request) \Rightarrow ❸ CGI บนผู้ส่งเซิร์ฟเวอร์ทำงานโดยการส่งคำสั่งไปค้นหาข้อมูลที่อยู่ในฐานข้อมูล \Rightarrow ❹ ค้นหาข้อมูลในฐานข้อมูล ถ้าค้นพบจะส่งรายເຍີດຂອງข้อมูลທັງหมดກັບໄປຍັງ CGI แต่ถ้าไม่พบจะแสดงผลว่า “ไม่พบข้อมูล” \Rightarrow ❺ CGI จะจัดเรียงรูปแบบของข้อมูลอย่างเป็นระบบ เช่น รูปร่างหน้าตาของการแสดงผล จำนวนข้อมูลในการแสดงผล รูปแบบของตัวอักษร (Font) ขนาดของตัวอักษร เป็นต้น เพื่อส่งกลับไปให้ผู้เรียกใช้งานผ่านโปรโตคอล HTTP (Reply) \Rightarrow ❻ เว็บไคลเอนต์รับข้อมูลที่ส่งมาจาก CGI และแสดงผล

2. การติดตั้งเว็บเซิร์ฟเวอร์และการปรับแต่ง (Web Server & Configuration)

ก่อนจะเขียนโปรแกรมเพื่อให้ทำหน้าที่เป็น CGI จำเป็นต้องมีโปรแกรมเว็บเซิร์ฟเวอร์ติดตั้งอยู่บนเครื่องก่อน ซึ่งโปรแกรมเว็บเซิร์ฟเวอร์ที่นิยมใช้งานอย่างมากในปัจจุบันคือ Apache webserver มีขั้นตอนการติดตั้งดังนี้

ติดตั้ง Apache Webserver

- ดาวน์โหลด Apache webserver ได้ที่ URL: <http://httpd.apache.org/>
- คลิกดาวน์โหลดแฟ้มชื่อ httpd-2.x.xx-win32-x86-openssl-x.x.x.msi (สำหรับวินโดวส์)
- ดับเบิลคลิกแฟ้มดังกล่าว จะปรากฏเมนู Apache HTTP Server 2.0 – Installation Wizard เลือก \Rightarrow Next > เลือก I accept the terms in the license agreement เลือก Next > \Rightarrow Read the first เลือก Next >
- เมนู Server Information ในส่วน Network Domain ให้ผู้ใช้ป้อนโดเมนตามที่ต้องการ เช่น servertest.com, ในส่วน Server Name เช่น www.servertest.com, ในส่วน Administrator's Email Address เช่น webmaster@servertest.com เลือก Next >
- เมนู Setup Type เลือก Typical เลือก Next >
- เมนู Destination Folder เลือก Next >
- เมนู Ready to Install the Program เลือก Install > และรอจนกว่าโปรแกรมติดตั้งเสร็จ \Rightarrow คลิกเลือก finish
- ทดสอบการทำงานของเว็บเซิร์ฟเวอร์โดยเรียกไปยัง URL: localhost บนเว็บไคลเอนต์ ถ้าผลลัพธ์ที่ได้แสดงในรูปที่ 14.4 แสดงว่าเว็บเซิร์ฟเวอร์พร้อมใช้งานแล้ว



รูปที่ 14.4 แสดงเว็บเซิร์ฟเวอร์พร้อมทำงาน

ปรับแต่งเว็บเซิร์ฟเวอร์เพื่อสนับสนุนการทำงานของ CGI

โดยปกติ Apache Webserver สนับสนุนการทำงานของ CGI อยู่แล้ว แต่จำเป็นต้องปรับแต่งเว็บเซิร์ฟเวอร์ให้รองรับการทำงานกับไฟล์ Python ลีกน้อย โดยมีขั้นตอนดังนี้

1. เปิดแฟ้มคอนฟิกกูเรชัน (Configuration file) ด้วยโปรแกรม Notepad เพื่อปรับแต่งเว็บเซิร์ฟเวอร์ชื่อ **httpd.conf** ซึ่งอยู่ที่ไดเรคทรอรี่ C:\Program Files (x86)\Apache Group\Apache2\conf
2. ให้ค้นหา Keyword คำว่า Options Indexes FollowSymLinks ให้เพิ่มคำว่า 'ExecCGI' ดังนี้
Options Indexes FollowSymLinks ExecCGI
3. ค้นหา Keyword คำว่า #AddHandler cgi-script .cgi และลบสัญลักษณ์ # ออกดังนี้
AddHandler cgi-script .cgi .py
4. บันทึกแฟ้ม httpd.conf และรีเมต์ตันเว็บเซิร์ฟเวอร์ใหม่ (restart apache)



Note: ถ้าไม่สามารถบันทึกแฟ้มข้อมูล httpd.conf ได้ อาจสืบเนื่องมาจากผู้ใช้งานไม่มีสิทธิ์ในการเขียนแฟ้มข้อมูล ให้ผู้ใช้งานคลิกขวาที่ไฟล์เดอรี่ C:\Program Files (x86)\Apache Group\Apache2\htdocs ⇒ Security ⇒ edit ⇒ Users ⇒ Permission for Users ⇒ Allow ให้เป็น Full control และกดปุ่ม OK จนกว่าจะเสร็จสิ้น

ทดสอบการทำงานของ Python CGI Script

1. เขียนโปรแกรมด้วยภาษา Python ชื่อว่า hello.py ดังนี้

```
hello.py
⇒1 #!/Python34/python
2 print("Content-type: text/html")
3 print("")
```

```

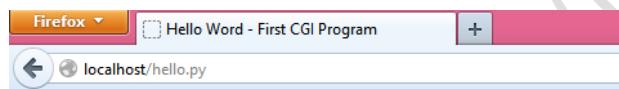
4   print("<html><head>")
5   print("")
6   print("</head><body>")
7   print("Hello from Python.")
8   print("</body></html>")

```

บรรทัดที่ 1 ของโปรแกรม hello.py เป็นการประกาศว่าให้เว็บเซิร์ฟเวอร์เรียกใช้ไฟล์ชื่อ hello.py ในไดเรคทรอรี่ /Python34/python ซึ่งหมายถึง C:\Python34\python.exe นั่นเอง (ถ้าติดตั้งไฟล์ Python ไว้ในไดเรคทรอรี่อื่นๆ ที่แตกต่างจากที่แสดงไว้ ให้ผู้เขียนโปรแกรมเปลี่ยนไปยังไดเรคทรอรี่ที่ติดตั้งไฟล์ Python ไว้) จากนั้นทำการบันทึก hello.py ไว้ในไดเรคทรอรี่ C:\Program Files (x86)\Apache Group\Apache2\htdocs

2. ทดสอบการทำงานของไฟล์ hello.py โดยเปิดเว็บไคเบนต์และป้อนที่อยู่ดังนี้ URL:

<http://localhost/hello.py> ซึ่งจะได้ผลลัพธ์ดังรูปที่ 14.5



Hello Word! This is my first CGI program

รูปที่ 14.5 แสดงผลการทำงานของโปรแกรม hello.py

แต่ถ้าเขียนโปรแกรมผิดไวยกรณ์จะทำให้เกิดข้อผิดพลาดดังตัวอย่างที่ 14.6



Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.
Please contact the server administrator, webmaster@servertest.com and inform them of the time the error occurred.
More information about this error may be available in the server error log.

Apache/2.0.65 (Win32) Server at localhost Port 80

รูปที่ 14.6 แสดงผลลัพธ์เมื่อไวยกรณ์ผิดพลาด

3. การเขียนโปรแกรม CGI สคริปต์

ก่อนที่จะเขียนโปรแกรม CGI สคริปต์ที่ซับซ้อนขึ้น ผู้เขียนจำเป็นต้องเข้าใจกระบวนการทำงานของโพรโทคอล HTTP เสียก่อน โดยเรียนรู้จากโปรแกรมตัวอย่างที่ 14_1 ดังนี้

Program Example 14_1: Python CGI Script

```

1  #!/Python34/python
2  print ("Content-type:text/html\r\n\r\n")
3  print ('<html>')
4  print ('<head>')

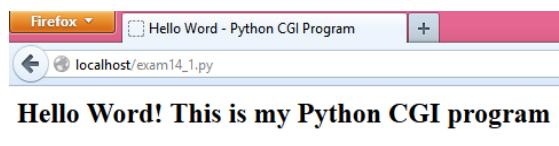
```

```

5  print ('<title>Hello Word - Python CGI Program</title>')
6  print ('</head>')
7  print ('<body>')
8  print ('<h2>Hello Word! This is my Python CGI program</h2>')
9  print ('</body>')
10 print ('</html>')

```

ทดสอบการทำงานโดยเปิดเว็บเบราว์เซอร์และป้อน URL ดังนี้คือ http://localhost/exam14_1.py จะได้ผลลัพธ์ดังรูป



จากตัวอย่างโปรแกรมที่ 14_1 แสดงการเขียนโปรแกรม CGI สคริปต์เพื่อพิมพ์ข้อความว่า 'Hello Word - Python CGI Program' ลงบนส่วน Title bar ของเว็บเบราว์เซอร์ และพิมพ์ข้อความว่า 'Hello Word! This is my Python CGI program' โดยมีขนาดตัวอักษรใหญ่เท่ากับ H2 ลงบนส่วนพื้นที่ทำงาน (Workspace) ของเว็บเบราว์เซอร์ ดังรูปด้านบน

บรรทัดที่ 2 โปรแกรมสั่งพิมพ์ "Content-type:text/html\r\n\r\n" ซึ่งหมายถึงรหัสส่วนหัวของโปรโตคอล HTTP (HTTP Header) มีรูปแบบดังนี้

Field_Name : Field_Content

เช่น Content-type:text/html\r\n\r\n โดย Field_Name เท่ากับ Content-type และ Field_Content เท่ากับ text/html\r\n\r\n สำหรับรหัสส่วนหัวที่จำเป็นในการเขียนโปรแกรมกับ CGI สคริปต์มีหลายชนิดดังตารางที่ 14.1

ตารางที่ 14.1 แสดงรหัสส่วนหัวของ HTTP โปรโตคอล

รหัสส่วนหัว (Header)	คำอธิบาย
Content-type:	บอกให้เว็บบราวเซอร์รู้ว่าเนื้อหาที่เซิร์ฟเวอร์ส่งกลับมาเป็นแฟ้มประเภทอะไร เช่น Content-type:text/html เป็นเอกสาร html, image/jpeg เป็นภาพ และ application/pdf เป็นประเภท pdf เป็นต้น
Expires: Date	ใช้สำหรับบอกวันและเวลาหมดอายุของเอกสาร ให้กับเว็บบราวเซอร์เพื่อทำการอ่านข้อมูลใหม่ มีรูปแบบ เช่น 01 Jan 1998 12:00:00 GMT.
Location: URL	เป็น URL ที่จะถูกส่งกลับไปยังผู้ร้องขอ และสามารถใช้รหัสดังกล่าวเปลี่ยนทิศทาง (Redirect) ไปยังแฟ้มอื่นๆ ได้
Last-modified: Date	เวลาล่าสุดที่แก้ไขหรือปรับปรุงแฟ้ม
Content-length: N	ความยาวของข้อมูล (ไบต์) ที่จะส่งกลับไปยังเว็บบราวเซอร์ เพื่อคำนวณเวลาในการดาวน์โหลดแฟ้ม

Set-Cookie: String	กำหนดค่าคุกกี้ (Cookie) ในลักษณะสตริง
--------------------	---------------------------------------

ตัวแปรสภาพแวดล้อมที่ใช้กับ CGI สคริปต์ (Environment Variables)

ตัวแปรสภาพแวดล้อมใช้สำหรับเก็บข้อมูลต่างๆ ขณะที่ CGI สคริปต์กำลังประมวลผล โดยเมื่อผู้ใช้กดปุ่ม submit ในฟอร์ม (Form) ที่ผ่านของเว็บไคลเอนต์ ข้อมูลจะถูกส่งมายังเว็บเซิร์ฟเวอร์ โดยสิ่งแรกที่เว็บเซิร์ฟเวอร์จะกระทำการหลังจากที่ได้รับข้อมูลที่ส่งมาจากฟอร์มนั้นคือ การกำหนดค่าในตัวแปรสภาพแวดล้อมให้กับ CGI สคริปต์เพื่อทำงาน ตัวแปรสภาพแวดล้อมที่สำคัญแสดงในตารางที่ 4.2

ตารางที่ 4.2 แสดงตัวแปรสภาพแวดล้อมที่ใช้กับ CGI สคริปต์

รหัสส่วนหัว (Header)	คำอธิบาย
CONTENT_TYPE	บอกประเภทของข้อมูลตามแบบ MIME ใช้เมื่อไคลเอนต์ส่งแฟ้มข้อมูลมาพร้อมกับฟอร์มไปยังเซิร์ฟเวอร์ เช่น การอัพโหลดแฟ้ม เป็นต้น
CONTENT_LENGTH	ความยาวของข้อมูลแบบสอบถามที่ส่งไปยังเว็บเซิร์ฟเวอร์ เมื่อใช้กับเมธอด POST
HTTP_COOKIE	ส่งค่ากลับเป็นเซ็ตของคุกกี้ที่มีลักษณะเป็นคู่คือ คีย์กับข้อมูล
HTTP_USER_AGENT	ชื่อและเวอร์ชันของไคลเอนต์ เช่น Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US)
HTTP_ACCEPT	บอกประเภทของข้อมูลตามแบบ MIME ที่เว็บбраузரรับได้
PATH_INFO	ตำแหน่งที่อยู่ของ CGI สคริปต์
QUERY_STRING	เป็นตัวแปรที่ใช้เก็บข้อมูลทั้งหมดจากแบบฟอร์มที่ถูกส่งให้กับเว็บเซิร์ฟเวอร์โดยวิธี GET
REMOTE_ADDR	หมายเลขไอพีของเครื่องไคลเอนต์ที่ร้องขอมา�ังเว็บเซิร์ฟเวอร์
REMOTE_HOST	เก็บชื่อเต็ม (fully qualified name) ของเครื่องไคลเอนต์ที่ร้องขอมา�ังเว็บเซิร์ฟเวอร์
REQUEST_METHOD	บอกวิธีการที่ร้องขอมา�ังเว็บเซิร์ฟเวอร์ เช่น POST หรือ GET
SCRIPT_FILENAME	เก็บที่อยู่แบบเต็ม (full path) ของ CGI สคริปต์ เช่น C:\Program Files (x86)\Apache Group\Apache2\htdocs\hello.py
SCRIPT_NAME	เก็บชื่อของ CGI สคริปต์
SERVER_NAME	เก็บชื่อหรือไอพีของเว็บเซิร์ฟเวอร์ ถ้าไม่กำหนดจะมีค่าเป็น 127.0.0.1
SERVER_SOFTWARE	เก็บชื่อและรุ่นของซอฟต์แวร์ที่ทำงานอยู่บนเว็บเซิร์ฟเวอร์
AUTH_TYPE	บอกวิธีการพิสูจน์ตัวตนผู้ใช้งาน

ตัวอย่างโปรแกรมที่ 14_2 แสดงรายการของตัวแปรสภาพแวดล้อมของ CGI สคริปต์

Program Example 14_2: Environment Variables for CGI

```

1 #!/Python34/python
⇒2 import os
⇒3 print("Content-type: text/html\r\n\r\n")
⇒4 print("<font size=+1>Environment</font><br />")
⇒5 for param in os.environ.keys():
⇒6     print("<br><br>%20s</b>: %s<br />"%(param, os.environ[param]))

```



OUTPUT

```

Environment
SERVER_SOFTWARE: Apache/2.0.65 (Win32)
REQUEST_URI: /exam14_2.py
SCRIPT_FILENAME: C:/Program Files (x86)/Apache Group/Apache2/htdocs/exam14_2.py
REMOTE_PORT: 50294
SERVER_PROTOCOL: HTTP/1.1
REQUEST_METHOD: GET
SERVER_ADMIN: webmaster@servertest.com
REMOTE_ADDR: 127.0.0.1
HTTP_ACCEPT:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
HTTP_USER_AGENT: Mozilla/5.0 (Windows NT 6.2; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.117
Safari/537.36
PATH:
C:\Python33\;C:\Python34\;C:\Python34\Scripts;C:\Windows\system32;
C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;
HTTP_CONNECTION: keep-alive
HTTP_HOST: localhost
SERVER_ADDR: 127.0.0.1
SCRIPT_NAME: /exam14_2.py
SERVER_PORT: 80
HTTP_ACCEPT_ENCODING: gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE: th-TH,th;q=0.8
SYSTEMROOT: C:\Windows
WINDIR: C:\Windows
PATHEXT:
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY<r>
QUERY_STRING:
DOCUMENT_ROOT: C:/Program Files (x86)/Apache Group/Apache2/htdocs
COMSPEC: C:\Windows\system32\cmd.exe
HTTP_CACHE_CONTROL: max-age=0
SERVER_NAME: localhost
GATEWAY_INTERFACE: CGI/1.1
SERVER_SIGNATURE:
Apache/2.0.65 (Win32) Server at localhost Port 80

```

ในตัวอย่างโปรแกรมที่ 14_2 บรรทัดที่ 2 โปรแกรมนำเข้าโนดูล os เพื่อใช้สำหรับดึงข้อมูลต่างๆ ของระบบมาแสดงผล บรรทัดที่ 3 เว็บเซิร์ฟเวอร์บอกให้เว็บไคลเอนต์ทราบว่าเนื้อหาที่ส่งมาให้เป็นเอกสารชนิด HTML ("Content-type: text/html") พร้อมกับขึ้นบรรทัดใหม่จำนวน 2 บรรทัด (/n/n) บรรทัดที่ 4 เว็บเซิร์ฟเวอร์สั่งให้เว็บไคลเอนต์แสดงคำว่า "Environment" ที่มีขึ้นตอนต์เท่ากับ 1 บรรทัดที่ 5 เว็บเซิร์ฟเวอร์ดึงข้อมูลจากระบบอุปกรณ์แสดงผลโดยใช้คำสั่ง for ซึ่งข้อมูลดังกล่าว

ประกอบด้วยคู่ของคีย์ (Key) และข้อมูล (Value) เช่น คีย์คือ **SERVER_SOFTWARE** มีข้อมูลเท่ากับ "Apache/2.0.65 (Win32)" เป็นต้น จากนั้นบรรทัดที่ 6 โปรแกรมจะนำคีย์ที่ได้มาเพื่อดึงข้อมูลของระบบมาแสดงผลโดยใช้เมธอด `os.environ[param]` ซึ่ง param เท่ากับคีย์นั้นเอง โปรแกรมจะดึงข้อมูลมาแสดงผลที่ลักษณะกว้างข้อมูลจะหมด

เมธอด HTTP GET และ HTTP POST

วิธีการรวบรวมข้อมูลจากฟอร์มในฝั่งของไคลเอนต์เพื่อส่งให้กับ CGI ศรีปต์บนฝั่งเซิร์ฟเวอร์ นั้นโดยทั่วไปจะมีอยู่ 2 วิธีหลักๆ ขึ้นอยู่กับการกำหนดไว้ในฟอร์ม HTML ว่าจะส่งข้อมูลด้วยวิธีการใด

- เมธอด HTTP GET

เมธอด HTTP GET หมายถึง การส่งข้อมูลเรียงต่อไปทางด้านท้ายของ URL ซึ่งจะทำให้ตัวแปรและข้อมูลที่กรอกในฟอร์มแสดงต่อท้าย URL ที่ต้องการส่งข้อมูลไป โดยสามารถมองเห็นข้อมูลเหล่านั้นปรากฏในช่อง address ของบราวเซอร์ เช่น `http://localhost/get.php?name="Johny"&age=30`

- เมธอด HTTP POST

เมธอด HTTP POST เป็นวิธีการส่งข้อมูลอีกวิธีการหนึ่ง โดยไม่ส่งข้อมูลต่อท้ายไปร่วมกับ URL ทำให้มองไม่เห็นข้อมูลที่ส่ง สามารถรองรับปริมาณการส่งข้อมูลได้มากกว่าและมีความปลอดภัยมากกว่า ตัวอย่างโปรแกรมที่ 14_3 แสดงการใช้งานเมธอด HTTP GET พร้อมส่งข้อมูลไปพร้อมกับเมธอดดังกล่าว 2 ค่าคือ ชื่อและนามสกุลดังนี้

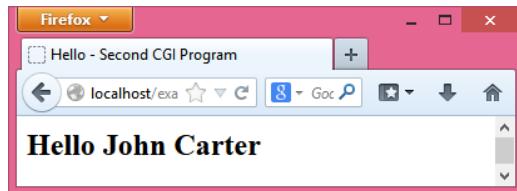
Program Example 14_3: HTTP GET Method

```

1  #!/Python34/python
2  # Import modules for CGI handling
3  import cgi, cgitb
4  # Create instance of FieldStorage
5  form = cgi.FieldStorage()
6  # Get data from fields
7  first_name = form.getvalue('first_name')
8  last_name = form.getvalue('last_name')
9  print ("Content-type:text/html\r\n\r\n")
10 print ("<html>")
11 print ("<head>")
12 print ("<title>Hello - Second CGI Program</title>")
13 print ("</head>")
14 print ("<body>")
15 print ("<h2>Hello %s %s</h2>" % (first_name, last_name))
16 print ("</body>")
17 print ("</html>")
```

ทดสอบการทำงานของโปรแกรมด้านบนโดยกรอกข้อมูลชื่อและนามสกุลในตำแหน่ง address ของบราวเซอร์ถัดจาก CGI ศรีปต์ ดังนี้

`http://localhost/exam14_3.py?first_name=John&last_name=Carter`



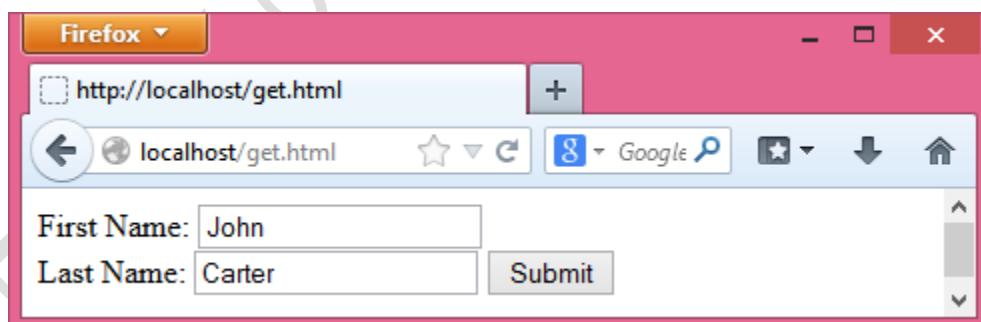
จากตัวอย่างโปรแกรมที่ 14_3 บรรทัดที่ 3 โปรแกรมนำเข้าโมดูล cgi และ cgitb เข้ามาเพื่อใช้ ประมวลผล CGI สคริปต์ บรรทัดที่ 5 โปรแกรมทำการสร้างการเชื่อมโยงไปยังฟอร์ม HTML โดยเรียกใช้ เมธอด cgi.FieldStorage() บรรทัดที่ 7 โปรแกรมทำการดึงข้อมูลชื่อ 'first_name' และ 'last_name' จาก ฟอร์มด้วยเมธอด form.getvalue() ค่าที่ได้จากเมธอดดังกล่าวจะถูกแสดงผลในบราวเซอร์ (บรรทัดที่ 15) โดยมีขนาดตัวอักษรเท่ากับ H2 และโปรแกรมทำการพิมพ์ title bar ด้วยข้อความว่า "Hello - Second CGI Program" ในบรรทัดที่ 12

เพื่อให้เข้าใจการทำงานของเมธอด HTTP GET เพิ่มเติม ในตัวอย่างโปรแกรมชื่อ get.html แสดง การสร้างฟอร์ม HTML ที่ประกอบไปด้วยฟิวต์ 'first_name' และ 'last_name' เช่นเดียวกับตัวอย่างที่ผ่านมา แต่แตกต่างกันคือ โปรแกรมจะฝัง CGI สคริปต์ไว้ภายในแฟ้ม get.html สำหรับ CGI สคริปต์จะเก็บไว้ใน ไดเรกทรอรี่เดียวกับแฟ้ม get.html ดังนี้

Program Example get.html: Simple Form example with get

```
⇒1 <form action="exam14_3.py" method="get">
2   First Name: <input type="text" name="first_name"> <br/>
3
4   Last Name: <input type="text" name="last_name" />
5   <input type="submit" value="Submit" />
6 </form>
```

เรียกใช้งานโดยเปิดบราวเซอร์และใส่ข้อมูลในช่อง address ดังนี้คือ <http://localhost/get.html>



ให้ผู้ใช้งานใส่ชื่อและนามสกุลลงในฟิวต์ First Name: เช่น 'John' และ Last Name: เช่น 'Carter' เมื่อกรอกข้อมูลเสร็จแล้วให้กดปุ่ม Submit เพื่อส่งฟอร์มไปประมวลผลบนเว็บเซิร์ฟเวอร์ ผลลัพธ์ที่ได้แสดงดังรูปด้านล่าง



จากตัวอย่างโปรแกรมชื่อ get.html แสดงตัวอย่างการสร้างฟอร์ม HTML เพื่อรับชื่อและนามสกุล เมื่อผู้ใช้กรอกข้อมูลครบแล้ว เมื่อกดปุ่ม Submit โปรแกรมจะเรียกเมธอด HTTP GET เพื่อส่งข้อมูลให้กับโปรแกรม exam14_3.py (บรรทัดที่ 1) ที่ได้อธิบายไว้ในตัวอย่างที่ผ่านมาทำงาน ผลลัพธ์ที่ได้คือ บรรทัดที่ 1 ของไฟล์ exam14_3.py จะแสดงชื่อและนามสกุลที่กรอกไว้ ดังตัวอย่างด้านบน

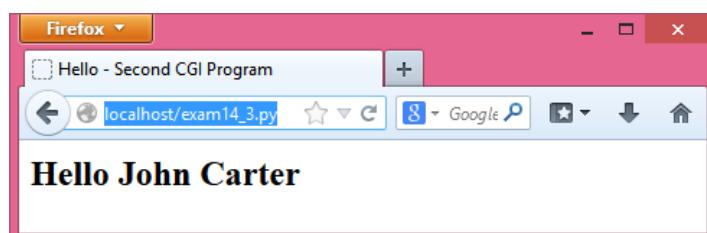
ในมุมมองด้านความปลอดภัย เมธอด HTTP GET จะมีความปลอดภัยน้อยกว่าเมธอด HTTP POST สิบเท่าเนื่องจาก HTTP GET จะแสดงข้อมูลแบบ clear text (ไม่มีการเข้ารหัสหรือซ่อนข้อมูล) ต่อท้ายไปกับ URL ทำให้เกิดจุดอ่อนในด้านความปลอดภัย แต่สำหรับเมธอด HTTP POST จะใช้เทคนิคการส่งข้อมูลที่แตกต่างกัน คือ POST จะรวบรวมข้อมูลจากฟอร์มเป็นแฟ้มข้อมูลชนิด text ก่อนแล้วจึงส่งแฟ้มดังกล่าวไปพร้อมกับ URL ต่อจากอักษร ? ทำให้ข้อมูลที่ส่งด้วยเมธอด POST มีความปลอดภัยมากกว่าเมธอด GET เพราะไม่มีการเปิดเผยเนื้อหาของข้อมูลโดยตรง ตัวอย่างโปรแกรมชื่อ post.html แสดงการใช้งานเมธอด HTTP POST โดยบันทึกแฟ้มชื่อ post.html และเรียกใช้ CGI ศรีริปต์ชื่อ exam14_3.py เมื่อันในตัวอย่างโปรแกรมที่ผ่านมา ดังนี้

Program Example post.html: Simple Form example with post

```

⇒1 <form action="exam14_3.py" method="post">
2   First Name: <input type="text" name="first_name"> <br/>
3
4   Last Name: <input type="text" name="last_name" />
5   <input type="submit" value="Submit" />
6 </form>
```

เรียกใช้งานโดยเปิดเบราว์เซอร์และใส่ข้อมูลในช่อง address ดังนี้คือ <http://localhost/post.html> ต่อจากนั้นให้กรอกข้อมูลชื่อและนามสกุลลงในฟอร์มและกดปุ่ม Submit ผลลัพธ์ที่ได้จะเห็นกับในตัวอย่างโปรแกรมที่ผ่านมา แต่แตกต่างกันคือ ในตัวอย่างโปรแกรม post.html จะไม่แสดงข้อมูลที่ผู้ใช้กรอกต่อท้ายกับ URL โดยตรง แต่จะถูกส่งเป็นแฟ้มข้อมูล text แทน ทำให้ข้อมูลมีความปลอดภัยเพิ่มขึ้น



- การส่งข้อมูลของ Checkbox ไปประมวลผลกับ CGI สคริปต์

Checkbox ถูกใช้เมื่อต้องการให้ผู้ใช้งานสามารถเลือกคำตอบได้มากกว่า 1 ตัวเลือก เช่น เลือกรายวิชาที่ต้องการเรียน หรือเลือกหนังสือที่ต้องการซื้อ เป็นต้น ตัวอย่างโปรแกรมชื่อ checkbox.html แสดงการสร้างฟอร์ม HTML โดยใช้ Checkbox สนับสนุนการทำงาน ภายในโปรแกรมเรียกใช้ CGI สคริปต์ชื่อ exam14_4.py โดยใช้เมธอด HTTP POST ดังนี้

Program Example checkbox.html: HTML form with Checkbox

```

⇒1 1 <form action="exam14_4.py" method="POST" target="_blank">
2   <input type="checkbox" name="maths" value="on" /> Maths
3   <input type="checkbox" name="physics" value="on" /> Physics
4   <input type="checkbox" name="computer" value="on" /> Computer
5   <input type="submit" value="Select Subject" />
6 </form>
```

สำหรับตัวอย่าง CGI สคริปต์ที่ฝังอยู่ในโปรแกรม checkbox.html ดังนี้

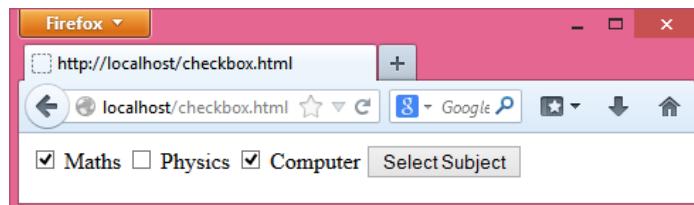
Program Example 14_4: CGI Script for processing Checkbox

```

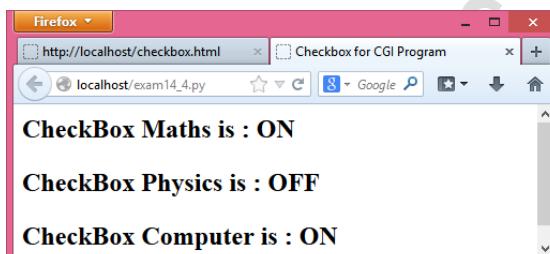
1  #!/Python34/python
2  import cgi, cgitb
3  form = cgi.FieldStorage()
⇒4 4 if form.getvalue('maths'):
5     math_flag = "ON"
6 else:
7     math_flag = "OFF"
⇒8 8 if form.getvalue('physics'):
9     physics_flag = "ON"
10 else:
11     physics_flag = "OFF"
⇒12 12 if form.getvalue('computer'):
13     computer_flag = "ON"
14 else:
15     computer_flag = "OFF"
16
17 print("Content-type:text/html\r\n\r\n")
18 print("<html>")
19 print("<head>")
20 print("<title>Checkbox for CGI Program</title>")
21 print("</head>")
22 print("<body>")
⇒23 23 print("<h2> CheckBox Maths is : %s</h2>" % math_flag)
⇒24 24 print("<h2> CheckBox Physics is : %s</h2>" % physics_flag)
⇒25 25 print("<h2> CheckBox Computer is : %s</h2>" % computer_flag)
26 print("</body>")
27 print("</html>")
```

จากตัวอย่างโปรแกรมที่ 14_4 บรรทัดที่ 4 โปรแกรมทำการตรวจสอบค่าในตัวแปร maths ว่าถูกเลือก (check) หรือไม่ ถ้า maths ถูกเลือก โปรแกรมจะกำหนดค่าตัวแปร math_flag เท่ากับ "ON" แต่ถ้าตัว

แปรดังกล่าวไม่ถูกเลือก math_flag จะถูกกำหนดค่าเป็น "OFF" เช่นเดียวกับโปรแกรมบรรทัดที่ 8 และ 12 ที่โปรแกรมดำเนินการกับตัวแปร physics และ computer ในบรรทัดที่ 23, 24 และ 25 โปรแกรมสั่งพิมพ์ข้อมูลที่เก็บในตัวแปร math_flag, physics_flag และ computer_flag ผลลัพธ์การทำงานของโปรแกรมแสดงในรูปด้านล่าง



จากรูปด้านบน แสดงผลการทำงานเมื่อเรียกโปรแกรม checkbox.html ให้ผู้ใช้งานคลิกเลือกวิชาใดวิชาหนึ่งหรือหลายวิชา ก็ได้ บน checkbox ดังรูปด้านบน และกดปุ่ม Select Subject ผลลัพธ์แสดงดังรูปด้านล่าง



- การส่งข้อมูลของ Radio button ไปประมวลผลกับ CGI สคริปต์

Radio button ถูกใช้กับงานที่ต้องการเลือกคำตอบได้เพียง 1 ตัวเลือกเท่านั้น เช่น เลือกปี พ.ศ. ที่เกิด หรือเลือกชื่อหน้า (นาย/นาง/นางสาว) เป็นต้น ตัวอย่างโปรแกรมชื่อ radio.html แสดงการสร้างฟอร์ม HTML โดยใช้ radio button สนับสนุนการทำงาน ภายใต้โปรแกรมเรียกใช้ CGI สคริปต์ชื่อ exam14_5.py ทำงานร่วมกับเมธอด HTTP POST ดังนี้

Program Example radio.html: HTML form with radio button

```

1 <form action="exam14_5.py" method="POST" target="_blank">
2   <input type="radio" name="subject" value="on" /> Maths
3   <input type="radio" name="subject" value="on" /> Physics
4   <input type="radio" name="subject" value="on" /> Computer
5   <input type="submit" value="Select Subject" />
6 </form>
```

สำหรับตัวอย่าง CGI สคริปต์ที่ฝังอยู่ในโปรแกรม radio.html ดังนี้

Program Example 14_5: CGI Script for processing radio button

```

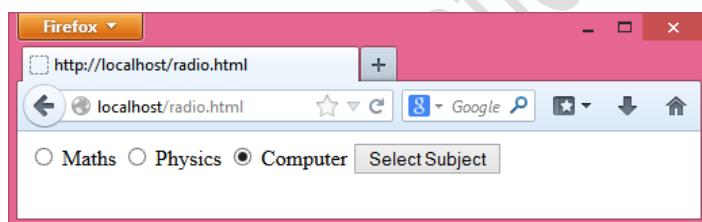
1 #!/Python34/python
2 import cgi, cgitb
3 form = cgi.FieldStorage()
```

```

⇒4 if form.getvalue('subject'):
5     subject = form.getvalue('subject')
6 else:
7     subject = "Not set"
8
9 print("Content-type:text/html\r\n\r\n")
10 print("<html>")
11 print("<head>")
12 print("<title>Radio for CGI Program</title>")
13 print("</head>")
14 print("<body>")
⇒15 print("<h2> Selected Subject is %s</h2>" % subject)
16 print("</body>")
17 print("</html>")

```

จากตัวอย่างโปรแกรมที่ 14_5 บรรทัดที่ 4 โปรแกรมทำการตรวจสอบค่าในตัวแปร subject ว่าถูกเลือก (check) หรือไม่ ถ้า subject ถูกเลือก โปรแกรมดึงค่าข้อมูลดังกล่าวมาเก็บไว้ในตัวแปร subject ซึ่งค่าที่ได้จะมี 2 สถานะเท่านั้นคือ ON และ OFF แต่ถ้าตัวแปรดังกล่าวไม่ถูกเลือก โปรแกรมจะกำหนดค่าให้กับตัวแปร subject เท่ากับ "Not set" ในบรรทัดที่ 15 โปรแกรมส่งพิมพ์ข้อมูลที่เก็บในตัวแปร subject ผลลัพธ์การทำงานของโปรแกรมแสดงในรูปต่อไปนี้



จากรูปด้านบนแสดงผลการทำงานเมื่อเรียกโปรแกรม radio.html ให้ผู้ใช้งานคลิกเลือกวิชาใดวิชาหนึ่งบน radio button ดังรูปด้านบน และกดปุ่ม Select Subject ผลลัพธ์แสดงดังรูปด้านล่าง



● การส่งข้อมูลของ Text Area ไปประมวลผลกับ CGI สคริปต์

Text Area ถูกใช้กับงานที่ต้องการประมวลผลข้อความที่มีจำนวนมากกว่า 1 บรรทัด เช่น ข้อมูลที่อยู่อาศัย หรือข้อความที่ต้องการโพสต์ในเว็บบอร์ด เป็นต้น ตัวอย่างโปรแกรมชื่อ textarea.html แสดงการสร้างฟอร์ม HTML โดยใช้ Text Area สนับสนุนการทำงาน ภายใต้โปรแกรมเรียกใช้ CGI สคริปต์ชื่อ exam14_6.py ดังนี้

Program Example textarea.html: HTML form with Text Area

```

⇒1 <form action="exam14_6.py" method="post" target="_blank">
2   <textarea name="textcontent" cols="40" rows="4">
3     This program for testing Text Area
4   </textarea>
5   <input type="submit" value="Submit" />
6 </form>

```

สำหรับตัวอย่าง CGI สคริปต์ที่ฝังอยู่ในโปรแกรม textarea.html ดังนี้

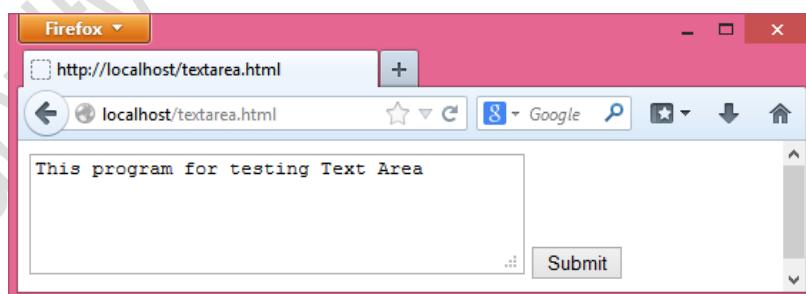
Program Example 14_6: CGI Script for processing Text Area

```

1  #!/Python34/python
2  import cgi, cgitb
3  form = cgi.FieldStorage()
⇒4 if form.getvalue('textcontent'):
5     text_content = form.getvalue('textcontent')
6 else:
7     text_content = "Not entered"
8
9 print("Content-type:text/html\r\n\r\n")
10 print("<html>")
11 print("<head>")
12 print("<title>Text Area for CGI Program</title>")
13 print("</head>")
14 print("<body>")
⇒15 print("<h2> Entered Text Content is %s</h2>" % text_content)
16 print("</body>")
17 print("</html>")

```

จากตัวอย่างโปรแกรมที่ 14_6 บรรทัดที่ 4 โปรแกรมทำการตรวจสอบค่าในตัวแปร textcontent ซึ่งเป็นชนิด Text Area ว่ามีการเขียนข้อความไว้หรือไม่ ถ้าผู้ใช้งานมีการเขียนข้อความไว้ โปรแกรมจะดึงค่าข้อความดังกล่าวมาเก็บไว้ในตัวแปร text_content และถ้าตัวแปรดังกล่าวไม่มีข้อความใดๆ โปรแกรมจะกำหนดค่าให้ตัวแปร text_content เท่ากับ "Not entered" แทน ในบรรทัดที่ 15 โปรแกรมสั่งพิมพ์ข้อมูลที่เก็บในตัวแปร text_content ผลลัพธ์ของการทำงานแสดงในรูปต่อไปนี้



จากรูปด้านบนแสดงผลการทำงานเมื่อเรียกโปรแกรม textarea.html ให้ผู้ใช้งานพิมพ์ข้อความใดๆ ก็ได้ลงบน Text Area ดังรูปด้านบน และกดปุ่ม Submit ผลลัพธ์แสดงดังรูปด้านล่าง



● การส่งข้อมูลของ Drop Down Box ไปประมวลผลกับ CGI สคริปต์

Drop Down Box ถูกใช้กับงานที่ต้องการเลือกคำตอบเพียง 1 คำตอบเท่านั้น จากคำตอบหลายๆ คำตอบที่มีอยู่ เช่น เลือกวิชาเรียน หรือเลือกชื่อจังหวัดที่อาศัย เป็นต้น ตัวอย่างโปรแกรมชื่อ dropdown.html แสดงการสร้างฟอร์ม HTML โดยใช้ Drop Down Box ภายในโปรแกรมเรียกใช้ CGI สคริปต์ชื่อ exam14_7.py ดังนี้

Program Example dropdown.html: HTML form with Drop Down Box

```

⇒1 <form action="exam14_7.py" method="post" target="_blank">
2   <select name="dropdown">
3     <option value="Maths" selected>Maths</option>
4     <option value="Physics">Physics</option>
5     <option value="Computer">Computer</option>
6   </select>
7   <input type="submit" value="Submit"/>
8 </form>
```

สำหรับตัวอย่าง CGI สคริปต์ที่ฝังอยู่ในโปรแกรม dropdown.html ดังนี้

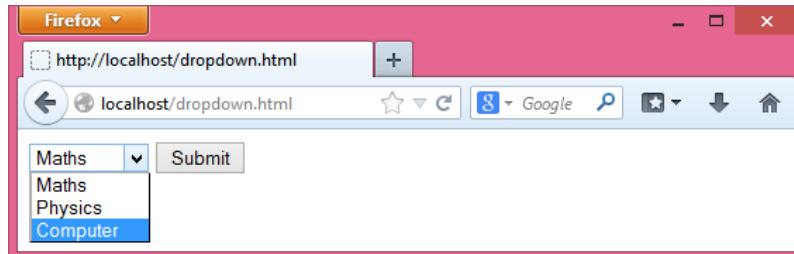
Program Example 14_7: CGI Script for processing Drop Down Box

```

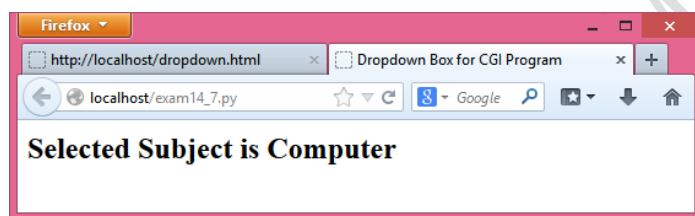
1  #!/Python34/python
2  import cgi, cgitb
3  form = cgi.FieldStorage()
⇒4  if form.getvalue('dropdown'):
5      subject = form.getvalue('dropdown')
6  else:
7      subject = "Not entered"
8
9  print("Content-type:text/html\r\n\r\n")
10 print("<html>")
11 print("<head>")
12 print("<title>Dropdown Box for CGI Program</title>")
13 print("</head>")
14 print("<body>")
⇒15 print("<h2> Selected Subject is %s</h2>" % subject)
16 print("</body>")
17 print("</html>")
```

จากตัวอย่างโปรแกรมที่ 14_7 บรรทัดที่ 4 โปรแกรมทำการตรวจสอบค่าในตัวแปร dropdown ซึ่งเป็นชนิด Drop Down Box ว่าได้ถูกเลือกหรือไม่ ถ้าผู้ใช้เลือกข้อมูลใน Drop Down ตัวได้ตัวหนึ่งไว้ โปรแกรมจะดึงค่าข้อความดังกล่าวมาเก็บไว้ในตัวแปร subject แต่ถ้าตัวแปรดังกล่าวไม่ได้เลือกข้อมูล

ได้ๆ ไว้ โปรแกรมจะกำหนดค่าให้ตัวแปร subject เท่ากับ "Not entered" แทน ในบรรทัดที่ 15
โปรแกรมสั่งพิมพ์ข้อมูลที่เก็บในตัวแปร subject ผลลัพธ์การทำงานของโปรแกรมแสดงในรูปต่อไปนี้



จากรูปด้านบนแสดงผลการทำงานเมื่อเรียกโปรแกรม dropdown.html ให้ผู้ใช้งานเลือกวิชาได้ๆ ใน Drop Down Box ดังรูปด้านบน และกดปุ่ม Submit ผลลัพธ์แสดงดังรูปด้านล่าง



● การใช้คุกกี้ (Cookie) ใน CGI สคริปต์

Cookie เป็นกลไกอย่างหนึ่งที่ช่วยให้ผู้พัฒนาโอมเพจสามารถใช้เก็บสถานะการใช้งานต่างๆ ของผู้เยี่ยมชมได้ โดยปกติแล้วโปรโทคอล HTTP เป็นโปรโทคอลที่มีลักษณะเป็น "stateless" คือไม่มีกลไกเกี่ยวกับการตรวจสอบสถานะต่างๆ ของผู้ใช้งาน

Cookies คือ ข้อมูลขนาดเล็กที่จะถูกส่งไปเก็บไว้ในบราวเซอร์ของผู้ใช้งาน เพื่อเก็บข้อมูลการเข้าเยี่ยมชม ช่วยให้สามารถติดตามว่าผู้เยี่ยมชมหน้าใดบ้าง เริ่มต้นจากหน้าไหน หรือบลลงด้วยหน้าไหน หรือบ่อยแค่ไหน เป็นต้น โดย Cookies ไม่ได้ใช้สำหรับตรวจสอบว่าผู้ใช้คือใคร

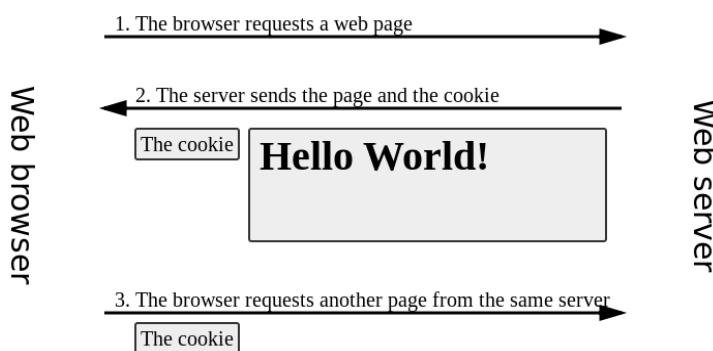
คุกกี้ทำงานอย่างไร

คุกกี้มีขั้นตอนการทำงานดังต่อไปนี้

- ผู้ใช้งาน (คนเข้าเยี่ยมชมเว็บไซต์) ส่งคำขอมาผ่านเบราว์เซอร์ฟิเวอร์ (โดยการเปิดบราวเซอร์เข้าหน้าเว็บปกติ) ดังรูปที่ 14.6
- เบราว์เซอร์จะทำการสร้างคุกกี้ขึ้นมาและส่งกลับไปให้กับเบราว์เซอร์ของผู้ใช้งาน
- เมื่อบราวเซอร์ในฝั่งของผู้ใช้งานยอมรับคุกกี้ บราวเซอร์จะสร้างแฟ้มขนาดเล็กในรูปแบบข้อความธรรมดา (plain text) เก็บไว้ในชาร์ดดิสก์ของผู้ใช้งาน ตัวอย่างข้อมูล

ของคุกกี้ เช่น .2o7.net TRUE/FALSE1234755376s_vi_nvnhg

- [CS]v4|3F09DC8800001DFF-A000A4A00000001|4032DDB1[CE]
4. เมื่อเวลาผ่านไป ผู้ใช้งานส่งคำร้องขอมา yang เซิร์ฟเวอร์อีก (เข้าเยี่ยมชมเป็นครั้งที่ 2) เว็บเซิร์ฟเวอร์จะสามารถจดจำได้ว่าผู้ใช้งานได้ทำกิจกรรมใดบ้างที่ผ่านมา



รูปที่ 14.6 แสดงการทำงานของ Cookie

รูปแบบของ Cookie ประกอบไปด้วย 5 ส่วนเชื่อมต่อกัน คือ

- Expires: วันหมดอายุของคุกกี้ เมื่อไม่ได้กำหนดข้อมูลในส่วนนี้ คุกกี้จะหมดอายุทันทีเมื่อผู้ใช้งานปิดเบราว์เซอร์
- Domain: ชื่อโดเมนของเว็บไซต์
- Path: ตำแหน่งที่อยู่สำหรับเก็บข้อมูลของคุกกี้ ถ้าไม่กำหนดไว้เบราว์เซอร์จะเก็บคุกกี้ไว้ในที่ใดๆ ก็ได้ในเครื่องผู้ใช้งาน
- Secure: เมื่อกำหนดค่าเท่ากับ "Secure" คุกกี้จะทำงานร่วมกับเว็บเซิร์ฟเวอร์แบบปลอดภัยเท่านั้น (Secure Server) ถ้าไม่กำหนดไว้คุกกี้จะไม่ทำงานในโหมดปลอดภัย
- Name=Value: คุกกี้ถูกกำหนดให้มีรูปแบบเป็นคู่ของคีย์กับข้อมูล (Key, Value)

การสร้าง Cookie

จากที่กล่าวมาแล้วว่าคุกกี้สร้างขึ้นจากผู้ของเว็บเซิร์ฟเวอร์ และส่งให้กับเบราว์เซอร์ของผู้ใช้งานผ่านโปรโตคอล HTTP (ซึ่งอยู่ในส่วนหัวของโปรโตคอล HTTP) จากตัวอย่างโปรแกรมที่ 14_8 แสดงการสร้างคุกกี้ โดยเรียกใช้เมธอด SimpleCookie() ดังนี้

Program Example 14_8: Creating Simple Cookie

```

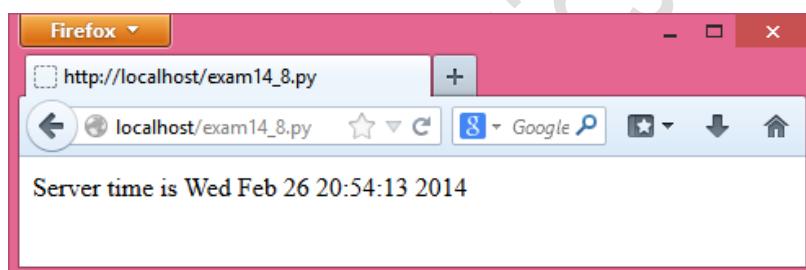
⇒1 #!/C:/Python27/python
⇒2 import time, Cookie
3
4 # Instantiate a SimpleCookie object
⇒5 cookie = Cookie.SimpleCookie()
6

```

```

7   # The SimpleCookie instance is a mapping
8   cookie['lastvisit'] = str(time.time())
9
10  # Output the HTTP message containing the cookie
11  print cookie
12  print 'Content-Type: text/html\n'
13  print '<html><body>'
14  print 'Server time is', time.asctime(time.localtime())
15  print '</body></html>'
```

จากตัวอย่างโปรแกรมที่ 14_8 สามารถเรียกใช้งานได้โดยพิมพ์ที่อยู่ในตำแหน่ง address ของบราวเซอร์ เท่ากับ `http://localhost/exam14_8.py` บรรทัดที่ 1 โปรแกรมนำเข้าตัวแปลงภาษาคือ Pythonเวอร์ชัน 2.7 เข้ามาทำงาน (เนื่องจากโมดูล Cookie ใช้งานได้เฉพาะ Python 2.7 สำหรับเวอร์ชัน 3 ขึ้นไปจะใช้เมธอด cookies() แทน) บรรทัดที่ 2 โปรแกรมนำเข้าโมดูล time และ Cookie เพื่อสนับสนุนการทำงานคุกคัก บรรทัดที่ 5 โปรแกรมเรียกเมธอด SimpleCookie() เพื่อสร้างคุกคักและเก็บไว้ในตัวแปรชื่อ cookie บรรทัดที่ 8 โปรแกรมกำหนดวันและเวลาล่าสุดที่เข้าเยี่ยมชมเว็บไซต์ บรรทัดที่ 11 โปรแกรมสั่งพิมพ์ค่า ของคุกคักออกจอภาพ ผลลัพธ์การทำงานของโปรแกรมแสดงในรูปต่อไปนี้



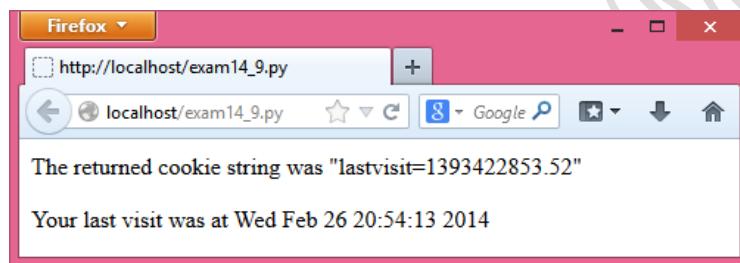
เมื่อต้องการแสดงค่าข้อมูลของคุกคักที่เก็บในเครื่องของผู้ใช้งาน สามารถเขียนโปรแกรมได้ดังนี้

Program Example 14_9: Retrieving Cookie

```

1  #!/usr/bin/python
2  import Cookie, os, time
3  cookie = Cookie.SimpleCookie()
4  print 'Content-Type: text/html\n'
5  print '<html><body>'
6  # The returned cookie is available in the os.environ dictionary
7  cookie_string = os.environ.get('HTTP_COOKIE')
8  # The first time the page is run there will be no cookies
9  if not cookie_string:
10    print '<p>First visit or cookies disabled</p>'
11  else: # Run the page twice to retrieve the cookie
12    print '<p>The returned cookie string was "' + cookie_string
13    + '"</p>'
14    # load() parses the cookie string
15    cookie.load(cookie_string)
16    # Use the value attribute of the cookie to get it
17    lastvisit = float(cookie['lastvisit'].value)
18    print '<p>Your last visit was at',
19    print time.asctime(time.localtime(lastvisit)), '</p>'
20  print '</body></html>'
```

ตัวอย่างโปรแกรมที่ 14_9 แสดงการดึงข้อมูลของคุกเกิลที่อยู่ในเครื่องมาแสดง บรรทัดที่ 1 โปรแกรมนำเข้าโมดูล Cookie, os และ time เข้ามาใช้งาน บรรทัดที่ 7 โปรแกรมเรียกใช้เมธอด environ.get() ที่อยู่ในโมดูล os โดยมีพารามิเตอร์ 1 ตัวคือ 'HTTP_COOKIE' ข้อมูลที่ส่งกลับมานาจากเมธอดดังกล่าวคือ ข้อมูลต่างๆ ที่เกี่ยวข้องกับคุกเกิล ซึ่งจะถูกเก็บไว้ในตัวแปร cookie_string บรรทัดที่ 9 โปรแกรมจะตรวจสอบข้อมูลในตัวแปร cookie_string ว่ามีข้อมูลอยู่หรือไม่ ถ้าไม่มี (ไม่ได้กำหนดค่า cookie ไว้) โปรแกรมจะพิมพ์ข้อความว่า 'First visit or cookies disabled' แต่ถ้าตัวแปรดังกล่าวมีข้อมูล โปรแกรมจะพิมพ์ข้อความว่า 'The returned cookie string was' ตามด้วยค่าข้อมูลของคุกเกิลที่เก็บอยู่ในตัวแปร cookie_string บรรทัดที่ 14 โปรแกรมทำการแปลงข้อมูลคุกเกิลให้ผู้ใช้งานสามารถอ่านได้ง่ายโดยใช้เมธอด cookie.load() บรรทัดที่ 16 และ 17 โปรแกรมได้นำเอาข้อมูลของ lastvisit ที่แปลงแล้วมาแสดงผลทางจอภาพดังรูปข้างล่าง



- การอัพโหลดแฟ้มด้วย CGI สคริปต์

การสร้างฟอร์ม HTML สำหรับอัพโหลดแฟ้มข้อมูล จะต้องใช้แอ็ตทริบิวต์ multipart/form-data ดังตัวอย่างโปรแกรมชื่อ upload.html

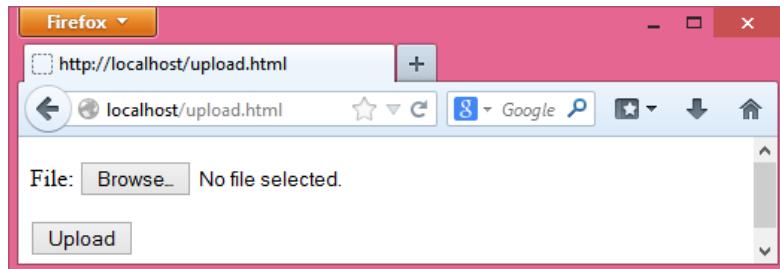
Program Example upload.html: HTML form with upload file
--

```

⇒1 <html>
2   <body>
3     <form enctype="multipart/form-data" action="exam14_10.py"
method="post">
4       <p>File: <input type="file" name="filename" /></p>
5       <p><input type="submit" value="Upload" /></p>
6     </form>
7   </body>
8 </html>

```

ผลลัพธ์การทำงานของโปรแกรม upload.html แสดงดังรูปด้านล่าง ให้ผู้ใช้เลือกแฟ้มข้อมูลที่ต้องการอัพโหลดโดยคลิกเลือก Browse... และกดปุ่ม Upload



สำหรับตัวอย่าง CGI สคริปต์ที่ทำหน้าที่อัพโหลดไฟล์ที่ฝังอยู่ในโปรแกรม upload.html ดังนี้

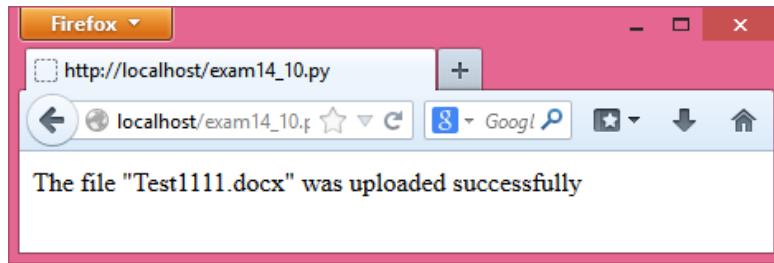
Program Example 14_10: CGI Script for uploading file

```

1  #!/C:/Python27/python
2  import cgi, os
3  import cgitb; cgitb.enable()
4  form = cgi.FieldStorage()
5  # Get filename here.
6  fileitem = form['filename']
7
8  # Test if the file was uploaded
9  if fileitem.filename:
10     # strip leading path from file name to avoid
11     # directory traversal attacks
12     fn = os.path.basename(fileitem.filename)
13     open('/tmp/' + fn, 'wb').write(fileitem.file.read())
14     message = 'The file "' + fn + '" was uploaded successfully'
15 else:
16     message = 'No file was uploaded'
17
18 print """
19 Content-Type: text/html\n
20 <html>
21 <body>
22   <p>%s</p>
23 </body>
24 </html>
25 """ % (message, )

```

จากตัวอย่างโปรแกรมที่ 14_10 บรรทัดที่ 4 โปรแกรมทำการดึงข้อมูลจากฟอร์มเก็บไว้ในตัวแปรชื่อ form และสกัดข้อมูลจากฟอร์มชื่อว่า filename เก็บไว้ในตัวแปร fileitem บรรทัดที่ 9 โปรแกรมตรวจสอบตัวแปร fileitem.filename ว่ามีชื่อแฟ้มที่ต้องการอัพโหลดหรือไม่ ถ้าผู้ใช้เลือกแฟ้มที่ต้องการอัพโหลดแล้ว โปรแกรมจะทำการกำหนดที่อยู่แบบเต็ม (บรรทัดที่ 12) และทำการเปิดแฟ้มเพื่อเขียนข้อมูลลงในไดเรคทรอรี่ C:\tmp (บรรทัดที่ 13) และพิมพ์ข้อความว่า 'The file "ชื่อแฟ้ม" was uploaded successfully' แต่ถ้าผู้ใช้ไม่ได้เลือกแฟ้มที่ต้องการอัพโหลดโปรแกรมจะพิมพ์ข้อความว่า 'No file was uploaded' (บรรทัดที่ 15) ดังตัวอย่างในรูปข้างล่าง



● การประยุกต์ CGI สคริปต์ทำงานหาร่วมกับฐานข้อมูล

โดยปกติเว็บไซต์ต่างๆ ที่เปิดให้บริการในปัจจุบัน เช่น ร้านค้าออนไลน์ ธนาคาร เว็บบอร์ด จะมีการจัดเก็บข้อมูลของลูกค้าเอาไว้ เพื่อความสะดวกในการทำธุกรรมในอนาคต เว็บไซต์ต่างๆ ที่ได้กล่าวมาแล้วนี้ นิยมใช้ฐานข้อมูลในการเก็บข้อมูล และใช้ภาษาสคริปต์ (VB, PHP, Perl, ASP script) ในการบริหารจัดการฐานข้อมูลอยู่เบื้องหลัง เช่น การเพิ่ม การลบ การปรับปรุง และการแสดงผลข้อมูลเป็นต้น ในหัวข้อนี้ผู้เขียนจะแนะนำวิธีการใช้งานไฟร่อนสคริปต์ในการบริหารจัดการฐานข้อมูลเช่นเดียวกับภาษาสคริปต์อื่นๆ โดยในตัวอย่างจะให้ลูกค้าป้อนข้อมูลชื่อ-สกุล อายุ เพศ และรายได้ผ่านทางฟอร์ม HTML จากนั้น HTML จะเรียกไฟร่อนสคริปต์เพื่อดำเนินการเพิ่มข้อมูลลงฐานข้อมูลชื่อ DBTest ในตารางชื่อว่า employee โปรแกรมชื่อ employeeForm.html มีหน้าที่รับข้อมูลของพนักงาน ประกอบไปด้วย ชื่อ (Name), นามสกุล (Surname), อายุ (Age), เพศ (Sex) และรายได้ (Income) งานนี้ทำการเพิ่มข้อมูลเหล่านี้ลงในฐานข้อมูลโดยใช้ CGI สคริปต์ ชื่อ exam14_11.py ดังนี้

Program Example employeeForm.html :

```

1 <html>
2 <head>
3 <title>User Registration</title>
4 </head>
5 <body>
6   <form action="exam14_11.py" method="post" target="_blank">
7     <h2>Employee Form</h2>
8     <h3>Name:&nbsp;<input maxlength="35" name="name" size="25" type="text"/></h3>
9     <h3>Surname:&nbsp;<input maxlength="35" name="surname" size="25" type="text"/></h3>
10    <h3>Age:&nbsp;<input maxlength="3" name="age" size="5" type="text"/></h3>
11    <h3>Sex: &nbsp;<strong>Male</strong><input name="male" type="radio"/>&nbsp;Female<input name="female" type="radio"/></h3>
12    <h3>Salary:&nbsp;<select name="income" size="1"><option value="20000">20000</option><option value="25000">25000</option><option value="30000">30000</option></select></h3>
13  </body>
14  <input type="submit" value="Submit" />
```

```
15  | </form>
16  | </html>
```

scrrip ท exm14_11.py แสดงได้ดังนี้

Program Example exm14_11.py:

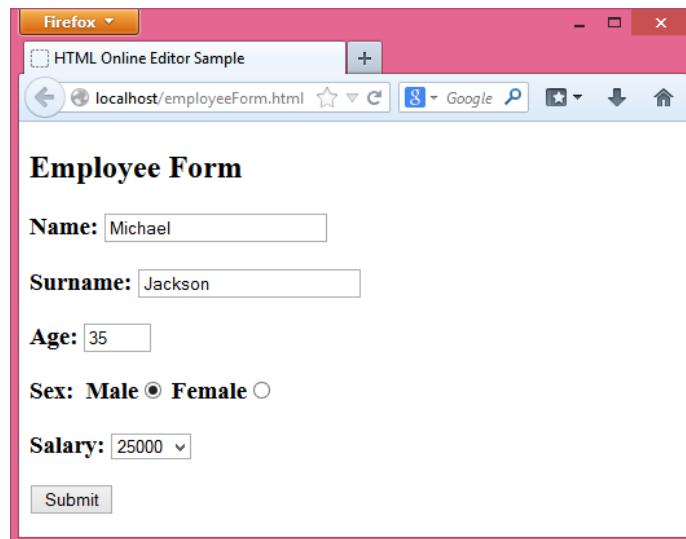
```
1  #!/Python27/python
2  import cgi, cgitb
3  import MySQLdb
4
5  form = cgi.FieldStorage()
6  if form.getvalue('name'):
7      name = form.getvalue('name')
8  else:
9      name = "NOT SET"
10
11 if form.getvalue('surname'):
12     surname = form.getvalue('surname')
13 else:
14     surname = "NOT SET"
15
16 if form.getvalue('age'):
17     age = int(form.getvalue('age'))
18 else:
19     age = 0
20
21 if form.getvalue('male'):
22     sex = 'M'
23 elif form.getvalue('female'):
24     sex = "F"
25
26 if form.getvalue('income'):
27     income = int(form.getvalue('income'))
28 else:
29     income = 20000
30
31 sql_insert = "INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES('%s', '%s', %d, '%c', %d)" %(name, surname, age, sex, income)
32 db = MySQLdb.connect("127.0.0.1", "root", "abc123", "DBTest" )
33 cursor = db.cursor()
34 try:
35     cursor.execute(sql_insert)
36     db.commit()
37 except:
38     db.rollback()
39 db.close()
40
41 print("Content-type:text/html\r\n\r\n")
42 print("<html>")
43 print("<head>")
44 print("<body>")
45 print("<h2>Name is %s</h2>" % name)
46 print("<h2>Surname is %s</h2>" % surname)
47 print("<h2>Age is %s</h2>" % age)
```

```

48 print("<h2>Sex is %s</h2>" % sex)
49 print("<h2>Income is %s</h2>" % income)
50 print("<h2>--- Inserted list ---</h2>")
51 print("<h2>%s</h2>" % sql_insert)
52 print("</body>")
53 print("</html>")

```

เมื่อเรียกใช้งาน employee.html ผลลัพธ์ที่ได้แสดงดังรูปด้านล่าง ให้ผู้ใช้กรอกข้อมูลจนครบแล้วกดปุ่ม Submit ดังนี้



Employee Form

Name:

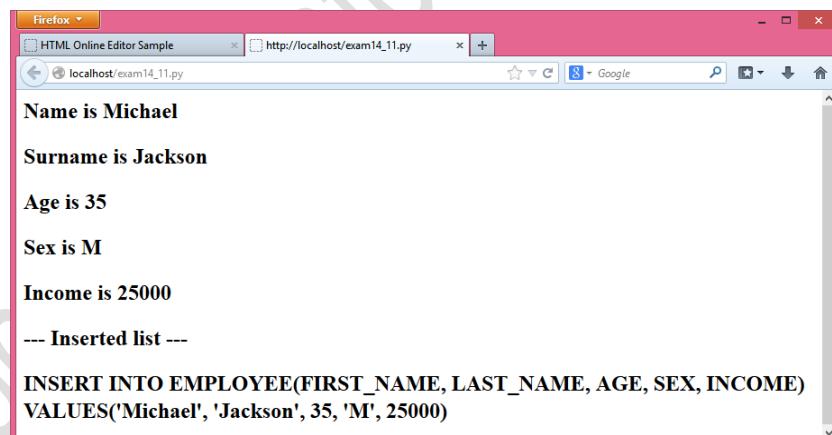
Surname:

Age:

Sex: Male Female

Salary:

เมื่อเพิ่มข้อมูลเรียบร้อยแล้ว โปรแกรมจะแสดงผลลัพธ์ดังนี้



Name is Michael
 Surname is Jackson
 Age is 35
 Sex is M
 Income is 25000
 --- Inserted list ---
 INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
 VALUES('Michael', 'Jackson', 35, 'M', 25000)

เมื่อต้องการแสดงผลข้อมูลที่ถูกจัดเก็บในฐานข้อมูล จะใช้ CGI ศิริปต์ชื่อ exam14_12.py ในการดึงข้อมูลมาแสดงผลดังนี้

Program Example exam14_12.py:

```

1 #!/Python27/python
2 import cgi, cgitb
3 import MySQLdb
4
5 sql_select = "SELECT * FROM EMPLOYEE"
6 db = MySQLdb.connect("127.0.0.1", "root", "abc123", "DBTest" )

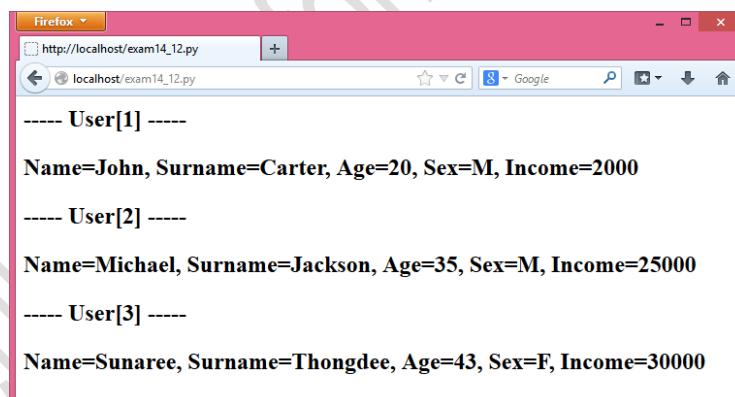
```

```

7  cursor = db.cursor()
8
9  print("Content-type:text/html\r\n\r\n")
10 print("<html>")
11 print("<head>")
12 print("<body>")
13 try:
14     count = 1
15     cursor.execute(sql_select)
16     results = cursor.fetchall()
17     for row in results:
18         name = row[0]
19         surname = row[1]
20         age = row[2]
21         sex = row[3]
22         income = row[4]
23         print("<h2>---- User[%d] ----</h2>" % count)
24         print("<h2>Name=%s, Surname=%s, Age=%d, Sex=%c," +
25               "Income=%d</h2>" % (name, surname, age, sex, income))
26         count += 1
27 except:
28     print ("Error: unable to fetch data")
29 db.close()
30
31 print("</body>")
32 print("</html>")

```

ผลลัพธ์ของโปรแกรม exam14_12.py แสดงดังนี้



จบบทที่ 14

บทที่ 15

การเขียนโปรแกรมกับมัลติ-thread และระบบเครือข่าย

(Multithread and Network Programming)

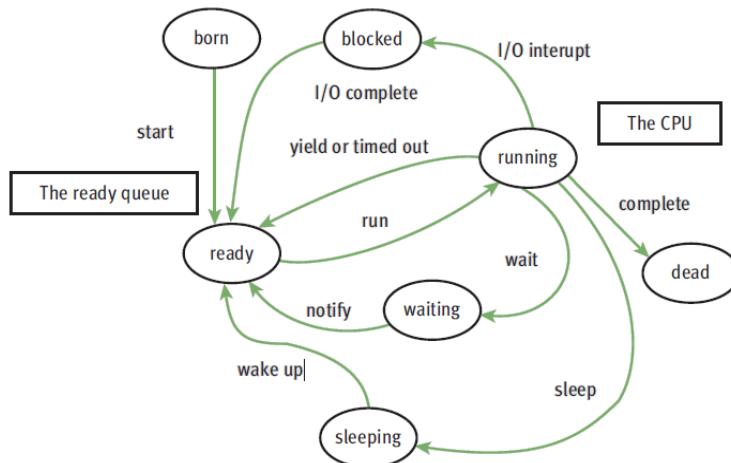


แอพพลิเคชันที่ใช้งานกันอยู่ในปัจจุบัน เช่น จดหมายอิเล็กทรอนิกส์ (Hotmail, Gmail, Yahoo), เว็บไซต์ให้บริการต่างๆ (Facebook, Instagram, Twisster, Ebay, Amazon, Sanook และอื่นๆ), เซิร์ชエンจิน (Google, Bring, Yahoo) เป็นต้น แอพพลิเคชันเหล่านี้ เป็นองหลังล้วนเกิดขึ้นมาจากการเขียนโปรแกรมโดยใช้เทคนิคของระบบเครือข่ายและมัลติ-thread ดึงเก็บทั้งสิ้น

1. เhread และプロเซส (Threads and Processes)

ระบบคอมพิวเตอร์สมัยใหม่อนุญาตให้มีผู้ใช้งานสามารถประมวลผลโปรแกรมได้มากกว่า 1 โปรแกรมพร้อมๆ กัน โดยเรียกเทคนิคนี้ว่า มัลติ-โปรเซสซิ่ง (Multi-Processing/Multi-Threading) ซึ่ง ส่งผลให้แอพพลิเคชันต่างๆ สามารถรองรับการเชื่อมต่อและใช้งานจากผู้ใช้งานได้มากกว่า 1 ช่องทาง ตัวอย่างเช่น ผู้ใช้งานสามารถใช้อีเมล์ (Gmail) ได้พร้อมๆ กันเป็นจำนวนมาก (1,000 users ขึ้นไป)

Process (โปรเซส) คือ โปรแกรมที่เริ่มต้นด้วยตัวตั้งแต่อยู่ในคิว (Queue) เพื่อรอการประมวลผลด้วย ซีพียู ไปจนถึงการยุติการทำงาน (Terminate) หรืออธิบายให้เข้าใจง่ายๆ คือ เมื่อผู้ใช้งานดับเบิลคลิกที่ไอคอนของโปรแกรมใดโปรแกรมหนึ่งขึ้นมาทำงาน และโปรแกรมดังกล่าวทำงานไปเรื่อยๆ จนกว่าจะปิด โปรแกรม เรียกโปรแกรมที่กำลังทำงานนั้นว่า โปรเซสหนึ่นเอง โดยช่วงชีวิตของโปรเซสมีอยู่หลายสถานะ (Process State) ดังรูปที่ 15.1



รูปที่ 15.1 แสดงสถานะของprocressและthread

1. Ready queue (New) คือ สถานะการเลือกโปรแกรมเพื่อนำมาสร้างเป็นprocress โดยเลือกโปรแกรมดังกล่าวมาจากหน่วยความจำสำรอง (อาร์ดดิสก์) เมื่อโปรแกรมถูกเรียกเข้ามาแล้วระบบปฏิบัติการจะเปลี่ยนโปรแกรมเป็นprocress โดยการสร้าง Process Control Block (PCB) และจัดเตรียมพื้นที่หน่วยความจำในการประมวลผล เตรียมจัดตารางงานให้กับซีพียู และเตรียมอุปกรณ์ IO ต่างๆ ที่จำเป็นของprocressเพื่อให้ทำงานได้สำเร็จจากนั้นprocressจะถูกนำไปเข้าคิว (Job Queue) เพื่อรับประมวลผลในสถานะต่อไป
2. Ready คือ สถานะของprocressที่เตรียมเข้าไปใช้งานหน่วยประมวลผลกลาง (ซีพียู) ในสถานะนี้อาจจะมาจาก New หรือ Waiting หรือ Running ก็ได้ กระบวนการที่มาจาก New, Waiting, Running จะเข้าคิวเพื่อรอเข้าใช้หน่วยประมวลผลกลาง หรือเรียกว่า Ready Queue ก็ได้
3. Running คือ สถานะของprocressที่ได้เข้าไปใช้งานหน่วยประมวลผลกลาง ณ เวลาใดเวลาหนึ่ง โดยจะมีเพียง 1 procressเท่านั้นที่สามารถเข้าไปประมวลผลในซีพียูได้ เนื่องจากซีพียูทำงานด้วยความเร็วสูงมาก จึงไม่มีปัญหาในเรื่องการรอคิวย
4. Terminate หรือ dead เป็นสถานะprocressที่ได้รับการประมวลผลเสร็จเรียบร้อยแล้ว หรือprocressที่มีการทำงานที่ผิดปกติเกิดขึ้น
5. Waiting เป็นสถานะของprocressที่ได้เข้าไปประมวลผลกับซีพียูแล้ว และมีการเรียกใช้อุปกรณ์รับ - ส่งข้อมูลหรืออุปกรณ์ต่างๆ ซึ่งทรัพยากรเหล่านั้นยังไม่ว่าง หรือมีprocressอื่นกำลังใช้งานอยู่ procressเหล่านั้นจะเปลี่ยนจากสถานะ Running ไปเป็น Waiting หรือเรียกว่า Device Queue หรือ Waiting Queue ก็ได้

Thread (ธread) คือ ส่วนประกอบย่อยของprocress โดยปกติprocressที่มี 1 ธread เรียกว่า Single thread แต่ถ้า 1 procressมีหลายธreadจะเรียกว่ามัลติธread (Multithread) ตัวอย่างเช่น เมื่อผู้ใช้งานเปิดโปรแกรมเว็บบราวเซอร์ (Firefox, Crome, IE) พร้อมกับเรียกไปยังเว็บไซต์ที่ต้องการ เช่น

www.goole.com เรียกได้ว่าเป็น 1 โพรเซส แต่เมื่อบราเวอร์เริ่มทำงานอาจจะแบ่งเป็นหลายๆ เครด โดยเครดแรกทำหน้าที่โหลดรูปภาพจาก google Majority เครื่องผู้ใช้งาน เครดที่สองทำหน้าที่โหลดข้อมูล Text และเครดที่สามทำหน้าที่แสดงผลแอนนิเมชันบนหน้าบราเวอร์ของผู้ใช้งาน เป็นต้น

เพราะฉะนั้นก่อนการเขียนโปรแกรมกับระบบเครือข่ายมีความจำเป็นต้องเข้าใจการทำงานของ เครดก่อนเสมอ เพราะว่าความเข้าใจเรื่องเครดจะช่วยให้ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมกับ ระบบเครือข่ายได้ดีขึ้น

2. การเขียนโปรแกรมกับเครด

ไฟรอนได้จัดเตรียมโมดูลสำหรับบริหารจัดการกับเครดชื่อว่า `threading` ผู้เขียนโปรแกรมจะต้อง ขยาย (extend) คลาสชื่อว่า `Thread` เข้ามาทำงาน ดังตัวอย่างโปรแกรมที่ 15.1

Program Example 15.1: Thread class

```

⇒1 from threading import Thread
2
⇒3 class myThread(Thread):
⇒4     def __init__(self, name):
⇒5         Thread.__init__(self)
⇒6         self.name = name
7
⇒8     def run(self):
8         print("Hello, my name is %s\n" %self.getName())
9
10
⇒11 process1 = myThread("Thread 1")
11 process2 = myThread("Thread 2")
12 process3 = myThread("Thread 3")
⇒14 process1.start()
13 process2.start()
14 process3.start()
15
16

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

Hello, my name is Thread 1
Hello, my name is Thread 2
Hello, my name is Thread 3

```

จากตัวอย่างโปรแกรมที่ 15.1 แสดงการสร้างและใช้งานเครด โดยบรรทัดที่ 1 โปรแกรมนำเข้าคลาส `Thread` ที่อยู่ภายใต้โมดูล `threading` เข้ามาทำงาน บรรทัดที่ 3 สร้างคลาสใหม่ชื่อ `MyThread` โดยสืบ ทอดคุณสมบัติมาจากคลาส `Thread` บรรทัดที่ 4 สร้างคอนสตรักเตอร์ชื่อ `__init__` (สำหรับผู้อ่านที่ไม่มี ความรู้เกี่ยวกับ OOP สามารถอ่านเพิ่มเติมได้ในบทที่ 11) ทำหน้าที่กำหนดค่าแอ็ตทริบิวเบื้องต้นให้กับ คลาส `myThread` โดยบรรทัดที่ 5 ทำการโอลิเวอร์รีร์เด็มชุดคอนสตรักเตอร์ของคลาสแม่ (คลาส `Thread`) ซึ่งจะต้องทำเสมอ บรรทัดที่ 6 กำหนดค่าให้แอ็ตทริบิว `name` ของคลาส `myThread` จาก อาร์กิวเมนต์ที่ส่งมาจากการสร้างอินสแตนซ์ในบรรทัดที่ 11 – 13 เมื่อโปรแกรมเรียกใช้เมธอด `start()` กับเครดที่สร้างขึ้นจะส่งผลให้ไฟรอนค้นหาเมธอดที่มีชื่อว่า `run()` ทันที ซึ่งเมธอด `run()` ในโปรแกรมนี้จะ

ทำหน้าที่พิมพ์ข้อความว่า "Hello, my name is" ตามด้วยชื่อของ-thread เช่น "Hello, my name is Thread 1" บรรทัดที่ 11 – 13 โปรแกรมสร้างอินสแตนซ์ของ-thread ชื่อ process1, process2 และ process3 พร้อมกับส่งอาร์กิวเมนต์เป็นชื่อของ-thread คือ "Thread 1", "Thread 2" และ "Thread 3" ตามลำดับ บรรทัดที่ 14 – 16 สั่งเริ่มต้นการทำงานของ-thread ส่งผลให้โปรแกรมเรียกใช้เมธอด run() อัตโนมัติ

1. โมดูล Threading

จากที่กล่าวมาแล้วในตอนต้นว่าการเขียนโปรแกรมควบคุม-thread จำเป็นต้องนำเข้าโมดูล threading เข้ามาทำงานก่อนเสมอ ในหัวข้อนี้จะกล่าวถึงเมธอดต่างๆ ที่โมดูล threading เตรียมไว้ให้ดังนี้

- เมธอด `threading.activeCount()` คืนค่าจำนวนของ-thread ที่กำลังทำงานอยู่ในคิว
- เมธอด `threading.currentThread()` คืนค่าจำนวนของ-thread ที่กำลังประมวลผลกับซีพียู
- เมธอด `threading.enumerate()` คืนค่ารายการของ-thread ทั้งหมดที่ถูกสร้างขึ้นมาในระบบ ทั้งหมด

ภายในโมดูล threading มีคลาสที่สำคัญในการสร้างและใช้งาน-thread คือ คลาสชื่อ `Thread` ซึ่งใน คลาสดังกล่าวมีเมธอดที่สำคัญดังนี้

- เมธอด `run()` ทำหน้าที่เริ่มต้นการทำงานของ-thread ซึ่งผู้เขียนโปรแกรมจะต้องเพิ่ม โปรแกรมต้นฉบับที่ต้องการลงในส่วนนี้ เมธอดนี้จะทำงานทันทีหลังจากมีการเรียกเมธอด `start()`
- เมธอด `start()` ทำหน้าที่กระตุ้นให้-thread ทำงาน โดยจะมีความสัมพันธ์โดยตรงกับเมธอด `run()` ตามที่ได้กล่าวมาแล้ว
- เมธอด `join([time])` ทำหน้าที่กำหนดเวลาให้กับ-thread ก่อนที่-thread ดังกล่าวจะยุติการ ทำงาน หรือพูดง่ายๆ คือกำหนดเวลาเพื่อให้ยุติการทำงานนั่นเอง
- เมธอด `isAlive()` ทำหน้าที่ตรวจสอบว่า-thread ยังคงทำงานอยู่หรือไม่
- เมธอด `getName()` คืนค่าเป็นชื่อของ-thread
- เมธอด `setName()` กำหนดชื่อให้กับ-thread

สรุปขั้นตอนการสร้างและใช้งาน-thread จากโมดูล `threading` และคลาส `Thread` ดังนี้

- นำเข้าโมดูล `threading` เช่น
`from threading import Thread`

2. สร้างคลาสใหม่และต้องสืบทอดคุณสมบัติมาจากคลาส Thread เช่น


```
class myClass(Thread):
```
3. โอลเวอร์ไรร์ดดิ้ง (Overriding) เมธอดคอนสตรักเตอร์ของคลาส Thread คือ `__init__(self [,args])` ซึ่งจะมีอาร์กิวเมนต์หรือไมก็ได้ (option) เช่น


```
Thread.__init__(self)
```
4. โอลเวอร์ไรร์ดดิ้งเมธอด `run()` ของคลาส Thread โดยผู้เขียนโปรแกรมจะใช้พื้นที่ตรงส่วนนี้ในการเขียนโปรแกรมเพื่อให้-thread ทำงานตามที่ต้องการ


```
def run(self):
```

"Please writing your code in here!"

หลังจากที่ผู้เขียนโปรแกรมสร้างอินสแตนซ์หรืออ้อปเจกต์ของคลาสขึ้นมาแล้ว จะส่งให้-thread ทำงานโดยการเรียกผ่านเมธอด `start()` ซึ่งจะส่งผลให้โปรแกรมเข้าไปทำงานในเมธอด `run()` ทันที จากโปรแกรมตัวอย่างที่ 15.2 แสดงการสร้าง-thread จากโมดูล `threading` เพิ่มเติมจากตัวอย่างที่ 15.1

Program Example 15.2: Thread and sleep

```

⇒1 from threading import Thread
⇒2 import time
3
⇒4 class myThread (Thread): #Extending Thread class
⇒5     def __init__(self, threadID, name, counter):
⇒6         Thread.__init__(self) #Overriding __init__ method
⇒7         self.threadID = threadID
⇒8         self.name = name
⇒9         self.counter = counter
10
⇒11    def printTime(self, threadName, delay, counter):
12        while counter:
13            time.sleep(delay)
14            print ("%s: %s" % (threadName,
time.ctime(time.time())))
15            counter -= 1
16
⇒17    def run(self): #Overriding run method
18        print ("Starting " + self.name)
19        self.printTime(self.name, self.counter, 5)
20        print ("Exiting " + self.name)
21
22    # Create new threads
⇒23    thread1 = myThread(1, "Thread-1", 1)
⇒24    thread2 = myThread(2, "Thread-2", 2)
25
26    # Start new Threads
⇒27    thread1.start()
⇒28    thread2.start()
29

```

⇒30 | print ("Exiting Main Thread")

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 13 15:49:07 2014
Thread-2: Thu Mar 13 15:49:08 2014
Thread-1: Thu Mar 13 15:49:08 2014
Thread-1: Thu Mar 13 15:49:09 2014
Thread-2: Thu Mar 13 15:49:10 2014
Thread-1: Thu Mar 13 15:49:10 2014
Thread-1: Thu Mar 13 15:49:11 2014
Exiting Thread-1
Thread-2: Thu Mar 13 15:49:12 2014
Thread-2: Thu Mar 13 15:49:14 2014
Thread-2: Thu Mar 13 15:49:16 2014
Exiting Thread-2

```

ตัวอย่างโปรแกรมที่ 15.2 แสดงการสร้างและใช้งาน threading แบบหนึ่ง โดยเริ่มจากบรรทัดที่ 1 ทำการนำเข้าโมดูล threading เข้ามาใช้งาน บรรทัดที่ 2 นำเข้าโมดูลที่จัดการด้านเวลา (Time) มาทำงาน บรรทัดที่ 4 สร้างคลาสใหม่ชื่อ myThread โดยได้รับการสืบทอดคุณสมบัติเกี่ยวกับ threading มาจากคลาสแม่ ชื่อ Thread บรรทัดที่ 5 สร้างเมธอดคอนสตรักเตอร์ชื่อ __init__() ของคลาส myThread ซึ่งมีคุณสมบัติในการกำหนดค่าเริ่มต้นให้กับคลาส myThread เริ่มต้นจากการโอลูร์ไร์ดิ้งเมธอด __init__ ของคลาส Thread (บรรทัดที่ 6) ต่อจากนั้นกำหนดค่าให้กับแอ็ตทริบิวต์ threadID, name และ counter ในบรรทัดที่ 7, 8 และ 9 ตามลำดับ

บรรทัดที่ 11 สร้างเมธอดชื่อว่า printTime() ทำหน้าที่พิมพ์ชื่อและเวลาของแต่ละthreadที่กำลังทำงาน โดยมีพารามิเตอร์ 3 ตัวคือ threadName, delay และ counter ตามลำดับ (ส่วนพารามิเตอร์ self นั้นหมายถึงการอ้างคลาสของตัวเอง ถ้าไม่ใส่ไว้โปรแกรมจะเข้าใจว่าเป็นการเรียกเมธอดในคลาสแม่แทน จะทำให้โปรแกรมเกิดความผิดพลาด) เมธอด printTime() จะทำการห่วงเวลาการพิมพ์โดยเรียกใช้เมธอด time.sleep() ซึ่งเวลาที่ใช้ห่วงมีหน่วยเป็นวินาที (บรรทัดที่ 13) คำสั่งลูป while ในโปรแกรมจะทำงานไปเรื่อยๆ จนกว่าค่าในตัวแปร counter จะมีค่าเท่ากับ 0 บรรทัดที่ 14 โปรแกรมเรียกใช้เมธอด time.ctime(time.time()) ซึ่งจะส่งผลให้เวลาที่แสดงมีรูปแบบเป็น เดือน วัน วันที่ เวลา ปี เช่น Thu Mar 13 15:49:12 2014 บรรทัดที่ 17 โอลูร์ไร์ดิ้งเมธอด run() ภายในเมธอดดังกล่าว โปรแกรมจะสั่งพิมพ์ข้อความว่า "Starting " และตามด้วยชื่อของthread เมื่อพิมพ์ข้อความเรียบร้อยแล้ว จะเรียกใช้เมธอด printTime() ซึ่งต้องส่งอาร์กิวเมนต์ไปด้วย 3 ค่าคือ ชื่อ (name), เวลาที่ใช้ห่วงการทำงาน (delay) และตัวนับ (counter) การเรียกเมธอดดังกล่าวจะต้องขึ้นต้นด้วย keyword คำว่า self เสมอ เมื่อโปรแกรมกลับมาจากการทำงานในเมธอด printTime() และจะพิมพ์ข้อความว่า "Exiting" และตามด้วยชื่อของthread นั้นๆ (บรรทัดที่ 20)

บรรทัดที่ 23 และ 24 โปรแกรมสร้างอินสแตนซ์ของคลาส myThread ชี้อ้วว่า thread1 และ thread2 โดยส่งอาร์กิวเมนต์ให้กับคลาส myThread 3 ตัวคือ threadID, name และ counter เมื่อสร้าง อินสแตนซ์แล้ว โปรแกรมสั่งให้-thread ทำงานโดยเรียกใช้งานผ่านเมธอด start() ซึ่งจะทำให้เมธอด run() ในคลาส myThread ทำงานทันที ผลลัพธ์ที่ได้แสดงให้เห็นว่าโปรแกรมจะพิมพ์ข้อความสลับกันระหว่าง เธรด 1 และธรด 2 ไปเรื่อยๆ จนกว่าธรดจะเสร็จสิ้นการทำงาน



Note: การสร้างอินสแตนซ์จะส่งผลให้ เมธอด __init__() ในคลาสทำงานทันที ซึ่งจะถูกใช้ใน การเตรียมสภาพแวดล้อมให้พร้อมก่อนโปรแกรมทำงาน



Tips: การเรียกใช้ตัวแปร (แอตทริบิว) และเมธอด ใน OOP จะใช้คำนำหน้าด้วย self เช่นเดียวกัน โปรแกรมจะเข้าใจว่าเป็นการเรียกใช้แอตทริบิวหรือเมธอดจากคลาสแม่

2. ซิงโครไนซ์ธรด (Threads Synchronization)

จากที่กล่าวมาแล้วข้างต้นว่า Python สามารถสร้างธรดได้มากกว่า 1 ธรด ซึ่งธรดต่างๆ เหล่านี้มีความสามารถในการทำงานได้พร้อมๆ กัน หากธรดเหล่านี้ทำงานที่เป็นอิสระไม่ขึ้นต่อ กัน โดยสิ้นเชิง จะไม่เกิดปัญหาใดๆ แต่ในความเป็นจริงธรดเหล่านี้ต้องใช้ทรัพยากร่วมกันไม่มากก็น้อย ดังนั้นการทำงานของธรดใดธรดหนึ่ง อาจมีผลกระทบทางอ้อมกับธรดอื่นๆ โดยผ่านทางทรัพยากรที่ใช้งานร่วมกัน เพื่อมิให้ส่งผลกระทบและเกิดความเสียหายต่อระบบ จึงเป็นหน้าที่ของระบบปฏิบัติการที่จะต้องควบคุมหรือหลีกเลี่ยงการทำงานของแต่ละธรดที่มาเกี่ยวข้องกัน (Interaction) หน้าที่นี้เรียกว่า การเข้าจังหวะกันของธรด หรือการซิงโครไนซ์ธรด (Threads Synchronization)

โมดูล threading ได้จัดเตรียมกลไกสำหรับจัดการเรื่องซิงโครไนซ์ธรดให้กับผู้เขียนโปรแกรมไว้ ดังนี้คือ

- เมธอด threading.Lock() ทำหน้าที่ล็อกทรัพยากรที่ต้องการใช้งานร่วมกัน อ็อปเจ็กต์ที่คืนจากเมธอดดังกล่าวจะถูกใช้ในการควบคุมทรัพยากร
- acquire([blocking]) เป็นเมธอดที่ทำงานร่วมกับอ็อปเจ็กต์ที่สร้างจากเมธอด Lock() ทำหน้าที่บังคับให้ธรดทำงานในโหมดซิงโครไนซ์ โดยมีพารามิเตอร์ 1 ตัวคือ blocking ซึ่งเป็นอ็อปชัน (option) ถ้ากำหนดให้ blocking = 0 เมธอด acquire() จะคืนค่าเป็น 0 ทันทีเมื่อไม่สามารถล็อกทรัพยากรได้ และคืนค่าเป็น 1 เมื่อสามารถล็อกทรัพยากรได้ เมื่อ blocking ถูกกำหนดเป็น 1 โปรแกรมจะหยุดการร้องขอทรัพยากรที่ต้องการล็อก จากธรดอื่นๆ ไว้ และจะรอจนกว่าทรัพยากรที่ต้องการจะถูกปลดปล่อยเพื่อทำการล็อก สรุปสั้นๆ คือ ถ้ากำหนด blocking เป็น 1 โปรแกรมจะรอจนกว่าจะล็อกทรัพยากรได้

- เมธอด `release()` ทำหน้าที่ปลดปล่อยทรัพยากรที่โปรแกรมล็อคหรือครอบครองอยู่

การใช้งานชิงโครในชีร์เชรดแสดงในตัวอย่างโปรแกรมที่ 15.3

Program Example 15.3: Threads Synchronization

```

1   from threading import Thread
2   import threading
3   import time
4
5   class myThread (Thread):
6       def __init__(self, threadID, name, counter):
7           Thread.__init__(self)
8           self.threadID = threadID
9           self.name = name
10          self.counter = counter
11
12      def printTime(self, threadName, delay, counter):
13          while counter:
14              time.sleep(delay)
15              print ("%s: %s" % (threadName,
16                  time.ctime(time.time())))
17              counter -= 1
18
19      def run(self):
20          print ("Starting " + self.name)
21          # Get lock to synchronize threads
22          threadLock.acquire()
23          self.printTime(self.name, self.counter, 3)
24          # Free lock to release next thread
25          threadLock.release()
26
27          threadLock = threading.Lock()
28          threads = []
29
30          # Create new threads
31          thread1 = myThread(1, "Thread-1", 1)
32          thread2 = myThread(2, "Thread-2", 2)
33
34          # Start new Threads
35          thread1.start()
36          thread2.start()
37
38          # Add threads to thread list
39          threads.append(thread1)
40          threads.append(thread2)
41
42          # Wait for all threads to complete
43          for t in threads:
44              t.join()
45
46          print ("Exiting Main Thread")

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

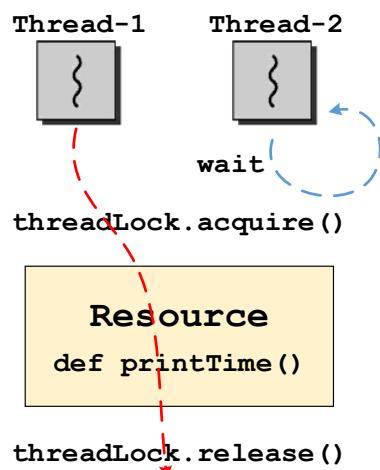
```

Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 13 18:13:57 2014
Thread-1: Thu Mar 13 18:13:58 2014
Thread-1: Thu Mar 13 18:13:59 2014
Thread-2: Thu Mar 13 18:14:01 2014
Thread-2: Thu Mar 13 18:14:03 2014
Thread-2: Thu Mar 13 18:14:05 2014
Exiting Main Thread

```

ตัวอย่างโปรแกรมที่ 15.3 แสดงการสร้างและใช้งานชิงโคร์ไนซ์เทรด บรรทัดที่ 5 และ 12 โปรแกรมสร้างคลาสชื่อ myThread และ เมธอด printTime() ตามลำดับ ซึ่งจะทำงานเหมือนโปรแกรมที่ 15.2 บรรทัดที่ 18 โปรแกรมทำการโอลิเวอร์เรตติ้งเมธอด run() จากคลาส Thread ซึ่งเมธอดดังกล่าวทำหน้าที่พิมพ์ชื่อเทรดและเวลาของเทรดนั้นๆ ออกจอภาพ ตามที่ได้กล่าวมาแล้วข้างต้นว่าเทรดจะทำงานเป็นอิสระต่อกัน ดังนั้น Thread-1 และ Thread-2 จะเข้าใช้งานเมธอด printTime() สลับกันไป ขึ้นอยู่กับว่าเมธอดใดจะเข้าถึงเมธอด printTime() ได้ก่อนกัน ดังนั้นในโปรแกรมนี้ได้ออกคำสั่ง (บรรทัดที่ 21) ให้เกิดการชิงโคร์ไนซ์เทรด ซึ่งหมายความว่าถ้าเทรดได้เทรดหนึ่งครอบครองเมธอด printTime() แล้ว เทรดนั้นๆ จะทำงานจนกว่าจะสำเร็จ โดยใช้เมธอด threadLock.acquire() เมื่อเทรดทำงานเสร็จแล้วจะปลดปล่อยเมธอด printTime() คืนสู่ระบบเพื่อให้เทรดอื่นๆ นำไปใช้งานต่อได้ โดยการเรียกใช้ threadLock.release() ดังในบรรทัดที่ 24 และแสดงการทำงานในรูปที่ 15.2

บรรทัดที่ 26 สร้างอินสแตนซ์ชื่อ threadLock เพื่อใช้สำหรับทำชิงโคร์ไนซ์เทรด บรรทัดที่ 27 สร้างตัวแปรลิสต์ชื่อ threads เพื่อใช้สำหรับเก็บอีบอปเจกต์ของเทรดทั้งหมดที่สร้างขึ้น บรรทัดที่ 30 โปรแกรมสร้างอินสแตนซ์จากคลาส myThread ชื่อ thread1 และ thread2 ตามลำดับ บรรทัดที่ 34 ออกคำสั่งกระตุ้นให้เทรดทำงานผ่านเมธอด start() บรรทัดที่ 38 เพิ่มเทรดลงในตัวแปรชนิดลิสต์ บรรทัดที่ 42 โปรแกรมใช้ลูป for ดึงออบเจกต์เทรดออกจากตัวแปร threads และใช้เมธอด join() เพื่อกำหนดให้เทรดทั้งหมดรอเพื่อจบการทำงานพร้อมๆ กัน



รูปที่ 15.2 แสดงการชิงโคร์ไนซ์เทรดระหว่าง Thread-1 และ Thread-2

3. การจัดลำดับความสำคัญของคิวสำหรับมัลติ-threaded (Multithreaded Priority Queue)

โมดูล Queue อนุญาตให้ผู้เขียนโปรแกรมสามารถสร้างคิวและกำหนดลำดับการทำงานของ thread ในคิวได้ด้วยตนเอง ซึ่งมีเมธอดให้ใช้งานดังนี้

- เมธอด get() ดึงข้อมูลออกจากคิว
- put() นำข้อมูลเข้าสู่คิว
- qsize() คืนค่าจำนวนของข้อมูลทั้งหมดที่อยู่ในคิว
- empty() ตรวจสอบว่าคิวว่างหรือไม่ ค่าที่ส่งคืนเท่ากับ 0 แสดงว่าคิวว่าง ค่าอื่นๆ แสดงว่าคิวไม่ว่าง
- full() ตรวจสอบว่าคิวเต็มหรือไม่ ค่าที่ส่งคืนเท่ากับ 0 แสดงว่าคิวเต็ม ค่าอื่นๆ แสดงว่าคิวไมเต็ม

การสร้างและใช้งานคิวแสดงในตัวอย่างโปรแกรมที่ 15.4

Program Example 15.4: Multithreaded Priority Queue

```

1   from threading import Thread
2   import threading
3   import queue
4   import time
5
6   exitFlag = 0
7
8   class myThread(Thread):
9       def __init__(self, threadID, name, q):
10          Thread.__init__(self)
11          self.threadID = threadID
12          self.name = name
13          self.q = q
14
15      def run(self):
16          print("Starting " + self.name + "\n")
17          self.process_data(self.name, self.q)
18          print("Exiting " + self.name + "\n")
19
20      def process_data(self, threadName, q):
21          while not exitFlag:
22              queueLock.acquire()
23              if not workQueue.empty():
24                  data = q.get()
25                  queueLock.release()
26                  print("%s processing %s\n" % (threadName,
27                      data))
28              else:
29                  queueLock.release()
                  time.sleep(1)

```

```

30
⇒31 threadList = ["Thread-1", "Thread-2", "Thread-3"]
⇒32 nameList = ["One", "Two", "Three", "Four", "Five"]
⇒33 queueLock = threading.Lock()
⇒34 workQueue = queue.Queue(10)
35
⇒36 threads = []
threadID = 1
37
38 # Create new threads
⇒39 for tName in threadList:
⇒40     thread = myThread(threadID, tName, workQueue)
⇒41     thread.start()
⇒42     threads.append(thread)
⇒43     threadID += 1
44
45 # Fill the queue
⇒46 queueLock.acquire()
⇒47 for word in nameList:
48     workQueue.put(word)
⇒49 queueLock.release()
50
51 # Wait for queue to empty
⇒52 while not workQueue.empty():
53     pass
54
55 # Notify threads it's time to exit
⇒56 exitFlag = 1
57
58 # Wait for all threads to complete
⇒59 for t in threads:
60     t.join()
61 print("Exiting Main Thread")

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

Starting Thread-1
Starting Thread-3
Starting Thread-2
Thread-1 processing One
Thread-3 processing Two
Thread-2 processing Three
Thread-1 processing Four
Thread-3 processing Five
Exiting Thread-2
Exiting Thread-1
Exiting Thread-3
Exiting Main Thread

```

ตัวอย่างโปรแกรมที่ 15.4 แสดงการสร้างและใช้งานคิวสำหรับมัลติ-thread บรรทัดที่ 3 โปรแกรมนำเข้าโมดูล queue เพื่อใช้สำหรับสร้างคิวในการบริหารจัดการเรตด บรรทัดที่ 6 กำหนดตัวแปร exitFlag เพื่อใช้สำหรับควบคุมเพื่อให้เรตดยุติการทำงาน ในเบื้องต้นกำหนดค่าให้กับตัวแปรดังกล่าวเท่ากับ 0 บรรทัดที่ 8 สร้างคลาสชื่อ myThread โดยมีพารามิเตอร์ 3 ตัวคือ threadID, Name และ q (queue)

บรรทัดที่ 15 – 18 โปรแกรมโอบเวอร์ไรร์ดิ้งเมธอด run() จากคลาส Thread ทำหน้าที่พิมพ์ข้อความว่า "Starting" ตามด้วยชื่อ-thread จากนั้นโปรแกรมเรียกใช้งานเมธอด process_data() เมธอดดังกล่าว ต้องการพารามิเตอร์ 2 ตัวคือ ชื่อ-thread (threadName) และคิว (Queue) เมื่อเมธอด process_data() ทำงานเสร็จแล้ว โปรแกรมจะพิมพ์ข้อความว่า "Exiting" ตามด้วยชื่อของ-thread

บรรทัดที่ 20 - 29 พังชันชื่อ process_data() ทำหน้าที่ดึง-thread ออกจากคิว (workQueue) และนำมายังผล เป็นที่ทราบกันดีแล้วว่า-thread จะแข่งขันกันเข้าใช้ทรัพยากร (ในที่นี้คือ workQueue) ดังนั้นพังชันดังกล่าวจึงใช้เมธอด queueLock.acquire() เพื่อร้องขอการเข้าใช้คิว ถ้า-thread ได้thread หนึ่ง ครอบครองคิว ได้สำเร็จ thread อื่นๆ จะต้องรอโดยจนกว่า-thread ที่ใช้งานคิวอยู่จะทำงานเสร็จก่อน เมื่อ-thread เข้าใช้งานคิวจะเรียกใช้เมธอด workQueue.empty() เพื่อตรวจสอบก่อนว่ามีข้อมูลอยู่ในคิวหรือไม่ ถ้าไม่มีโปรแกรมจะปลดปล่อยคิว แต่ถ้ามีข้อมูลอยู่ในคิวโปรแกรมจะใช้เมธอด q.get() เพื่อดึงข้อมูลในคิวออกมารажาน เมื่อ-thread ทำงานเสร็จแล้วจะปลดปล่อยคิว (workQueue) คืนให้กับ-thread อื่นๆ ได้ใช้งานบ้าง โดยใช้เมธอด queueLock.release()

บรรทัดที่ 31 สร้างชื่อ-thread เก็บไว้ในตัวแปร threadList โดยมีอยู่ 3 ชื่อคือ "Thread-1", "Thread-2" และ "Thread-3" บรรทัดที่ 32 กำหนดข้อมูลเพื่อกีบในคิวประกอบด้วย "One", "Two", "Three", "Four" และ "Five" บรรทัดที่ 33 สร้างอ็อปเจ็กต์ชื่อ queueLock ทำหน้าที่ล็อกและปลดปล่อยทรัพยากร ที่-thread แข่งขันกันใช้งาน (workQueue) บรรทัดที่ 34 สร้างคิวชื่อ workQueue ที่สามารถเก็บข้อมูลได้ 10 อ็อปเจ็กต์ บรรทัดที่ 36 สร้างตัวแปร threadID เพื่อใช้กำหนดจำนวน-thread ที่จะรันในโปรแกรม

บรรทัดที่ 39 – 43 โปรแกรมสร้างอินสแตนซ์ของคลาส myThread ซึ่งมีพารามิเตอร์ 3 ตัวคือ threadID, tName และ workQueue ในที่นี้โปรแกรมสร้าง-thread ขึ้นมา 3 เ.thread คือ ThreadID = 1 มีชื่อว่า Thread-1, ThreadID = 2 มีชื่อว่า Thread-2, ThreadID = 3 มีชื่อว่า Thread-3 จากนั้นโปรแกรมส่งให้-thread เหล่านี้ทำงานทันทีด้วยเมธอด start() และโปรแกรมทำการเพิ่ม-thread เหล่านี้ลงในลิสต์ของ-thread ชื่อ threads บรรทัดที่ 46 – 49 โปรแกรมทำการเพิ่มข้อมูลลงในคิว (workQueue) แต่เป็นไปได้ขณะเมื่อ โปรแกรมกำลังเพิ่มข้อมูลลงในคิวดังกล่าว ปรากฏว่า-thread ที่ส่งให้ทำงานไปแล้วอาจจะเข้ามาอ่านข้อมูล ในคิวพร้อมๆ กับการเพิ่มข้อมูลลงในคิวเช่นเดียวกัน ดังนั้นโปรแกรมจำเป็นต้องเรียกใช้งานเมธอด queueLock.acquire() เพื่อครอบครองคิวและทำการเพิ่มข้อมูลให้เสร็จก่อน เมื่อทำงานเสร็จแล้วจึงปลดปล่อยคิวด้วยเมธอด queueLock.release() เพื่อให้thread อื่นๆ สามารถใช้งานคิวต่อได้

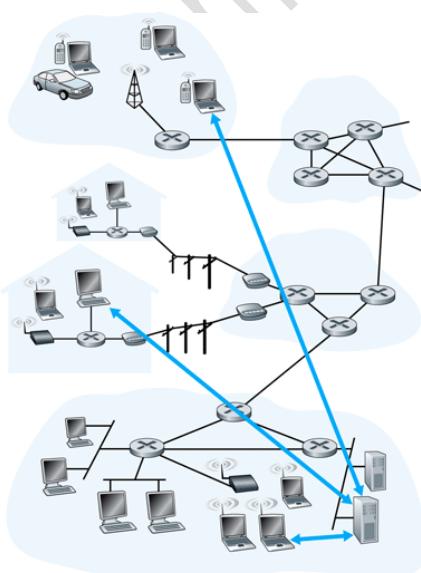
เมื่อโปรแกรมใส่ข้อมูลในคิวเสร็จเรียบร้อยแล้ว โปรแกรมจะทำงานต่อในบรรทัดที่ 53 (ในขณะนี้ thread อื่นๆ ก็ยังคงทำงานอยู่เช่นเดียวกัน) โปรแกรมจะประมวลผลคำสั่ง while โดยมีเงื่อนไขว่า โปรแกรมจะวนทำงานไปเรื่อยๆ จนกว่าคิว (workQueue) จะว่าง โปรแกรมจึงจะยุติการทำงานในลูป while นั้นหมายถึงว่า thread ต่างๆ ที่ทำงานอยู่ได้ดึงข้อมูลในคิวไปทำงานจนหมดแล้วนั่นเอง ลูป while

จึงยุติการทำงาน เมื่อคิวว่างแล้วส่งผลให้โปรแกรมหลุดจากลูป while (แต่ชีรด "Thread-1", "Thread-2", "Thread-3" ยังคงทำงานอยู่) และมาทำงานในบรรทัดที่ 56 ซึ่งเป็นการกำหนดค่าให้กับตัวแปรชื่อ exitFlag=1 ส่งผลให้ชีรดยุติการทำงาน (บรรทัดที่ 21)

บรรทัดที่ 59 โปรแกรมทำการวนลูปพร้อมกับใช้เมธอด join() เพื่อกำหนดให้ชีรดทุกๆ ตัวยุติการทำงานพร้อมกัน (ถูก Terminate พร้อมกัน)

3. การเขียนโปรแกรมเครือข่ายแบบคลients-เซิร์ฟเวอร์ (Clients-Servers)

สถานปัตยกรรมแบบคลients-เซิร์ฟเวอร์ ประกอบไปด้วย 2 ส่วนคือ ระบบคอมพิวเตอร์ที่มีประสิทธิภาพสูงทำหน้าที่เป็นเซิร์ฟเวอร์ ซึ่งจะให้บริการอยู่ตลอดเวลา และเครื่องคลients (Desktop PC, Notebook, smartphone, mobile-phone) ซึ่งจะทำหน้าที่ร้องขอการ คลients อาจจะเป็นเครื่องที่ทำงานตลอดเวลาหรือไม่ก็ได้และไม่สามารถสื่อสารกันเองได้ สำหรับเครื่องเซิร์ฟเวอร์จะต้องมีหมายเลขไอพีที่คงที่ (Fixed IP Address) รูปที่ 15.3 แสดงสถานปัตยกรรมแบบคลients-เซิร์ฟเวอร์ จากรูป เครื่องที่ทำหน้าที่เป็นเว็บเซิร์ฟเวอร์ (Web Server) จะคอยตอบสนองต่อการร้องขอจากเครื่องคลients ที่ทำงานด้วยโปรแกรมเว็บบราว์เซอร์ เป็นต้น



รูปที่ 15.3 แสดงสถานปัตยกรรมแบบคลients-เซิร์ฟเวอร์

1. หมายเลขไอพีแอดเดรส (IP Address)

คอมพิวเตอร์ที่เชื่อมต่อกับระบบเครือข่ายจำเป็นต้องมีหมายเลขไอพีแอดเดรสเพื่อใช้สำหรับติดต่อสื่อสารกัน โดยหมายเลขไอพีดังกล่าวต้องไม่ซ้ำกันเลย (Unique) สำหรับรูปแบบของไอพีแอดเดรสคือ $ddd.ddd.ddd.ddd$ โดย d คือเลขฐานสิบ และไอพีแอดเดรสแต่ละกลุ่ม

(ddd) จะมีค่าระหว่าง 0 - 255 ตัวอย่างเช่น 192.168.5.10, 136.112.195.77 เป็นต้น แต่การ
จดจำหมายเลขไอพีแอดเดรสเป็นเรื่องยากเนื่องจากมีความยาวมาก ดังนั้นจึงนิยมเรียกชื่อ^{เครื่องแทน} เช่น John-PC, Computer001 เป็นต้น สำหรับการเขียนโปรแกรมเพื่อแสดงข้อมูล
ไอพีแอดเดรสและชื่อเครื่อง ไฟรอนได้จัดเตรียมเครื่องมือไว้ให้คือ โมดูล socket ดังนี้

- เมธอด gethostname() แสดงชื่อเครื่องคอมพิวเตอร์ที่กำลังทำงานอยู่ ถ้า
คอมพิวเตอร์ไม่ได้กำหนดชื่อไว้ เมธอดดังกล่าวจะแสดงข้อความผิดพลาดอุปกรณ์
(exception) เช่น

```
>>> from socket import *
>>> gethostname()
'K-PC'
```
- เมธอด gethostbyname(ipName) แสดงหมายเลขไอพีแอดเดรสของเครื่องที่กำลัง
ทำงานอยู่ โดย ipName คือชื่อของเครื่องคอมพิวเตอร์ ถ้ากำหนดชื่อเครื่องไม่ถูก
โปรแกรมจะแสดงข้อผิดพลาดอุปกรณ์ เช่น

```
>>> gethostbyname(gethostname())
'192.168.1.104'
>>> gethostbyname('John')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    gethostbyname('John')
socket.gaierror: [Errno 11001] getaddrinfo failed
```

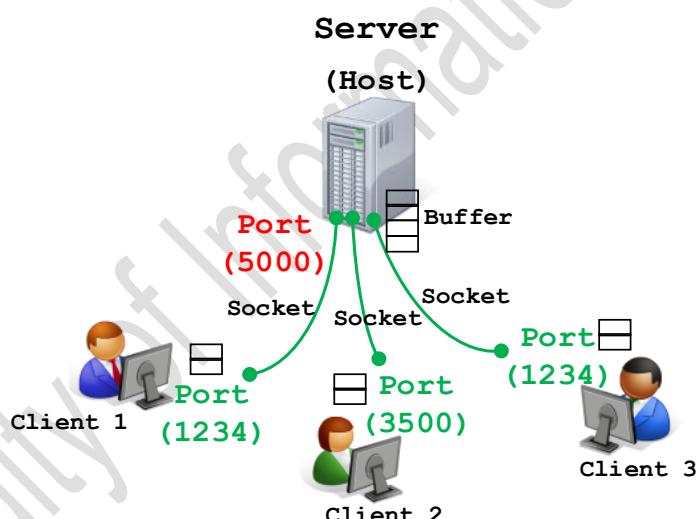
ชื่อเครื่องที่ใช้สำหรับทดสอบการทำงานของโปรแกรมเบื้องต้นเรียกว่า "localhost" มี
หมายเลขไอพีแอดเดรสนี้คือ 127.0.0.1 หรือเรียกว่า IP Loopback ก็ได้ หมายเลขไอพีดังกล่าวจะ
ไม่สามารถเชื่อมต่อกับระบบเครือข่ายได้ แต่มีหน้าที่สำหรับทดสอบการทำงานของแอพพลิเค
ชันที่พัฒนาขึ้นก่อนการนำไปใช้งานจริงเท่านั้น รูปแบบคำสั่งที่ใช้แสดงไอพี localhost คือ

```
>>> gethostbyname('localhost')
'127.0.0.1'
```

2. พอร์ต, เซิร์ฟเวอร์ และไคลเอ็นต์ (Ports, Servers and Clients)

พอร์ต คือ ช่องทางการเชื่อมต่อระหว่างเครื่องเซิร์ฟเวอร์และไคลเอ็นต์ เปรียบเสมือนทำเรือ^{ขนส่งสินค้า} ซึ่งเรือทุกๆ ลำจะต้องนำเรือสินค้ามาจอดที่ท่าเรือ (เปรียบเสมือนพอร์ต) ก่อนจากนั้นก็จึงทำ
การขนถ่ายสินค้า (เปรียบเสมือนข้อมูล) ได้ พอร์ตจะถูกกำหนดเป็นเลขจำนวนเต็มมีค่าระหว่าง 0 –
65535 โดยพอร์ตตั้งแต่ 0 – 1024 ถูกจดไว้เพื่อใช้งานพิเศษหรือเรียกว่า (Well known ports) เช่น ทำ
หน้าที่เป็นเว็บเซิร์ฟเวอร์ (พอร์ต 80), จดหมายอิเล็กทรอนิกส์ (25), โดเมนเนมเซิร์ฟเวอร์ (53) เป็นต้น
ดังนั้นหมายเลขพอร์ตตั้งแต่ 0 – 1024 จึงไม่ควรนำไปใช้ในการเขียนโปรแกรม สำหรับเครื่องไคลเอ็นต์

โดยปกติเมื่อมีการสื่อสารจะต้องใช้พอร์ตเดียวกับเครื่องเซิร์ฟเวอร์ แต่ระบบปฏิบัติการจะสุ่มเลือกขึ้นมาให้เองอัตโนมัติ การเขียนโปรแกรมเชื่อมต่อระหว่างเซิร์ฟเวอร์และไคลเอนต์จะใช้โมดูล Socket ในการทำงาน ซึ่ง Socket จะทำหน้าที่สร้างลิงค์เพื่อเชื่อมต่อระหว่างโปรเซสที่ทำงานอยู่ในฝั่งเซิร์ฟเวอร์กับไคลเอนต์ในลักษณะโอล์สต์ (Host) ต่อโอล์สต์ (เซิร์ฟเวอร์ 1 เครื่องกับไคลเอนต์ 1 เครื่อง) เรียกว่าการเปิดซ็อกเก็ต (Socket) หรือเปิดพอร์ตก็ได้ ผู้เขียนโปรแกรมสามารถเปิด Socket ได้มากกว่า 1 Socket บนหมายเลขพอร์ตเดียวกันได้ (เมื่อนำมาเรียกที่สามารถรองรับการจดเรือได้หลายลำ) โดยปกติแล้ว ความเร็วในการสื่อสารระหว่างไคลเอนต์และเซิร์ฟจะแตกต่างกัน เนื่องจากเซิร์ฟเวอร์จะมีประสิทธิภาพการทำงานที่สูงกว่าไคลเอนต์มาก และเซิร์ฟเวอร์จะต้องรองรับการร้องขอจากไคลเอนต์เป็นจำนวนมากในเวลาเดียวกัน ส่งผลให้การสื่อสารระหว่างเซิร์ฟเวอร์และไคลเอนต์อาจจะเกิดปัญหาในการรับส่งข้อมูลได้ ดังนั้นเพื่อแก้ปัญหาดังกล่าวจึงใช้บัฟเฟอร์ (Buffer) เข้ามาช่วยในการทำงาน ถ้าจะอธิบายให้ง่ายๆ บัฟเฟอร์คือ การจัดเรียงข้อมูลไว้ในพื้นที่ที่ได้จัดเตรียมไว้ก่อน เมื่อเครื่องคอมพิวเตอร์พร้อมทำงานแล้ว จะนำข้อมูลในบัฟเฟอร์ไปทำงานนั้นเอง จากรูปที่ 15.4 แสดงความสัมพันธ์ระหว่างเซิร์ฟเวอร์, ไคลเอนต์ พอร์ต, Sockets และ บัฟเฟอร์

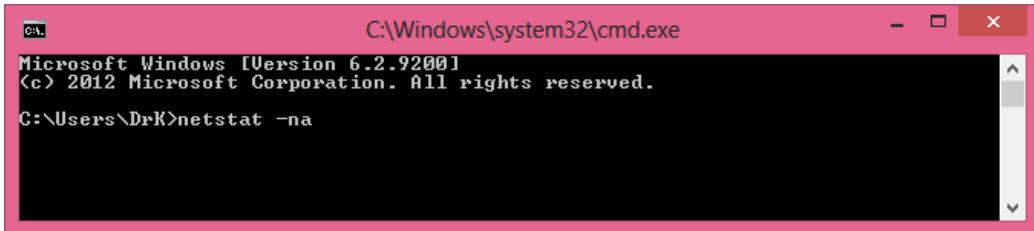


รูปที่ 15.4 แสดงความสัมพันธ์ระหว่าง Server, Clients, Port และ Sockets

3. เริ่มต้นการเขียนโปรแกรมเซิร์ฟเวอร์-ไคลเอนต์

โดยปกติแล้วเครื่องคอมพิวเตอร์ทุกๆ เครื่องจะเปิดพอร์ตที่ทำหน้าที่ต่างๆ ไว้มากมาย ผู้ใช้งานสามารถตรวจสอบการเปิดพอร์ตบนเครื่องของตนเองโดยใช้คำสั่ง netstate ใน Command rompt บน DOS ดังนี้

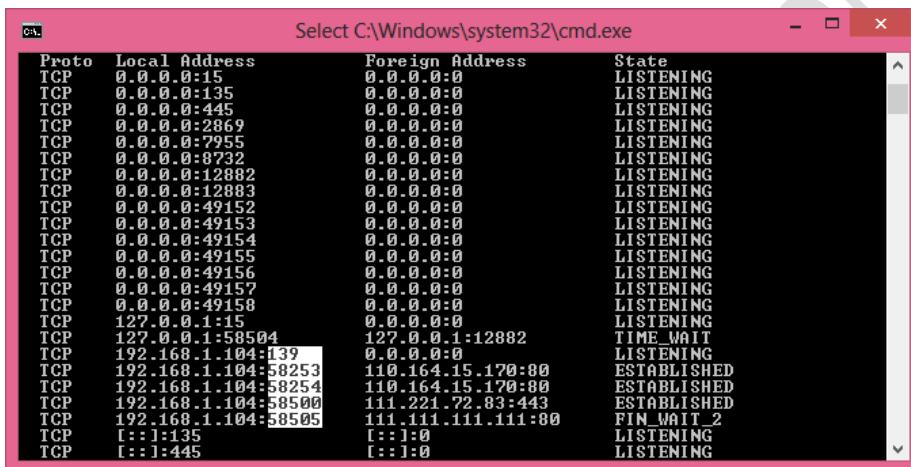
- เรียกโปรแกรม Command prompt บน DOS โดยคลิกที่ Start ⇒ run ⇒ พิมพ์คำสั่ง cmd และกดปุ่ม Enter ผลลัพธ์ที่ได้แสดงในรูปด้านล่าง



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\DrK>netstat -na
```

- พิมพ์คำสั่ง netstat -na ที่ Command prompt และกดปุ่ม Enter



Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:15	0.0.0.0:0	LISTENING
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:2869	0.0.0.0:0	LISTENING
TCP	0.0.0.0:7955	0.0.0.0:0	LISTENING
TCP	0.0.0.0:8732	0.0.0.0:0	LISTENING
TCP	0.0.0.0:12882	0.0.0.0:0	LISTENING
TCP	0.0.0.0:12883	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49152	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49153	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49154	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49155	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49156	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49157	0.0.0.0:0	LISTENING
TCP	0.0.0.0:49158	0.0.0.0:0	LISTENING
TCP	127.0.0.1:15	0.0.0.0:0	LISTENING
TCP	127.0.0.1:58504	127.0.0.1:12882	TIME_WAIT
TCP	192.168.1.104:139	0.0.0.0:0	LISTENING
TCP	192.168.1.104:58253	110.164.15.170:80	ESTABLISHED
TCP	192.168.1.104:58254	110.164.15.170:80	ESTABLISHED
TCP	192.168.1.104:58500	111.221.72.83:443	ESTABLISHED
TCP	192.168.1.104:58505	111.111.111.111:80	FIN_WAIT_2
TCP	[::]:135	[::]:0	LISTENING
TCP	[::]:445	[::]:0	LISTENING

จากรูปด้านบนตรงส่วนที่ Highlight ไว้คือหมายเลขพอร์ตต่างๆ ที่เครื่องคอมพิวเตอร์เปิดไว้

การเขียนโปรแกรมผู้ใช้เครื่องเดียว

ในตัวอย่างนี้ผู้เขียนจะแนะนำการเขียนโปรแกรมเพื่อดึงวันและเวลาจากเครื่องเซิร์ฟเวอร์มาแสดงผลในเครื่องไคลเอนต์ โดยอาศัยโมดูล Socket ช่วยในการเขียนโปรแกรม ซึ่งขั้นตอนการเขียนโปรแกรมผู้ใช้เครื่องเดียวดังนี้คือ

- สร้าง Socket (อ้อปเจ็กต์ซื้อกาเก็ต)
- เปิด Socket ด้วยพอร์ตที่สามารถใช้งานได้ (แนะนำให้ใช้พอร์ตตั้งแต่ 1,024 ขึ้นไป) ในเครื่องผู้ใช้เอง (localhost) ในที่นี่จะใช้พอร์ตหมายเลข 5,000
- ดึงวันและเวลาจากเครื่องเซิร์ฟเวอร์ผ่าน Socket
- แสดงผลวันและเวลาออกจอภาพ

สำหรับตัวอย่างโปรแกรมผู้ใช้เครื่องเดียวแสดงได้ดังนี้

Program Example 15.5: Easy Client Script (receive day/time)

```
1   ''' Client for obtaining the day and time from localhost.'''
2   from socket import *
```

```

3
⇒4 HOST = 'localhost'
⇒5 PORT = 5000
⇒6 BUFFER_SIZE = 1024
⇒7 ADDRESS = (HOST, PORT) #(127.0.0.1, 5000)
8
⇒9 server = socket(AF_INET, SOCK_STREAM) #Create a socket
⇒10 server.connect(ADDRESS) #Connect it to a host
⇒11 data_bytes = server.recv(BUFFER_SIZE) #Read a string from it
⇒12 dayAndTime = bytes.decode(data_bytes)
⇒13 print(dayAndTime)
⇒14 server.close() #Close the connection

```

จากตัวอย่างโปรแกรมที่ 15.5 และงโปรแกรมผังไคลเอ็นต์ ทำหน้าที่ร้องขอวันและเวลาบนเครื่องเซิร์ฟเวอร์มาแสดงผล บรรทัดที่ 2 นำเข้าโมดูล Socket โดยเลือกคลาส, เมธอดหั้งหมุด (*) ที่อยู่ในโมดูลดังกล่าวเข้ามาทำงาน บรรทัดที่ 4 กำหนดค่าให้กับตัวแปร HOST เท่ากับ 'localhost' (127.0.0.1) บรรทัดที่ 5 กำหนดค่าหมายเลขพอร์ต (PORT) เท่ากับ 5000 ซึ่งหมายเลขพอร์ตดังกล่าวอยู่บนผังเซิร์ฟเวอร์ บรรทัดที่ 6 กำหนดขนาดของบัฟเฟอร์เท่ากับ 1024 โดยมีหน่วยเป็นไบต์ (Bytes) บรรทัดที่ 7 กำหนดค่าให้กับตัวแปร ADDRESS มีค่าໄอพีเท่ากับ 127.0.0.1 และหมายเลขพอร์ตเท่ากับ 5000 (ในตัวอย่างนี้ เซิร์ฟเวอร์และไคลเอ็นต์จะทำงานอยู่ในเครื่องเดียวกัน) บรรทัดที่ 9 สร้างชั้อกเก็ตคอนเน็คชันด้วยคลาส socket โดยคลาสดังกล่าวต้องการพารามิเตอร์ 2 ตัวคือ AF_INET ซึ่งเป็นมาตรฐานการเชื่อมต่อของโพรโทคอล TCP/IP และ SOCK_STREAM คือการเชื่อมต่อเม亥มีอนกับการสร้างท่อจำลอง (Pipe) ข้อมูลที่รับและส่ง โดยข้อมูลจะไหลในท่อดังกล่าวอย่างต่อเนื่อง บรรทัดที่ 10 ส่งให้เกิดการเชื่อมต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ด้วยหมายเลขໄอพีและพอร์ตที่กำหนดไว้ใน ADDRESS บรรทัดที่ 11 ไคลเอ็นต์ร้องขอข้อมูลจากเซิร์ฟเวอร์ด้วยเมธอด server.recv(BUFFER_SIZE) โดย BUFFER_SIZE คือขนาดของบัฟเฟอร์ที่ผังไคลเอ็นต์จะรับข้อมูลได้สูงสุดในแต่ละรอบ ค่าที่ส่งกลับมาจากเซิร์ฟเวอร์คือวันและเวลาเก็บไว้ในตัวแปรชื่อ data_bytes แต่เนื่องจากข้อมูลที่ส่งและรับของ Socket ในไฟรอน 3.0 ขึ้นไปจะใช้ข้อมูลชนิด byte แทนสตริง ดังนั้นจึงจำเป็นต้องทำการแปลงจาก byte เป็นสตริงเสียก่อนโดยใช้เมธอด bytes.decode() ดังแสดงในบรรทัดที่ 12 ข้อมูลที่แปลงแล้วจะเก็บไว้ในตัวแปรชื่อ dayAndTime ต่อจากนั้นบรรทัดที่ 13 พิมพ์ข้อมูลที่อยู่ในตัวแปร dayAndTime บรรทัดที่ 14 สิ่งที่โปรแกรมผังไคลเอ็นต์ยุติการเชื่อมต่อ ส่งผลให้เซิร์ฟเวอร์ปิดช่องทางการเชื่อมต่อจากเครื่องไคลเอ็นต์ดังกล่าว เมื่อเขียนโปรแกรมเสร็จแล้วทดสอบสิ่งที่รันโปรแกรมจะเกิดข้อผิดพลาดขึ้นคือ

Traceback (most recent call last):

```

File "C:/Python34/Ch15/Ch15_5.py", line 10, in <module>
    server.connect(ADDRESS)
ConnectionRefusedError: [WinError 10061] No connection could be made
because the target machine actively refused it

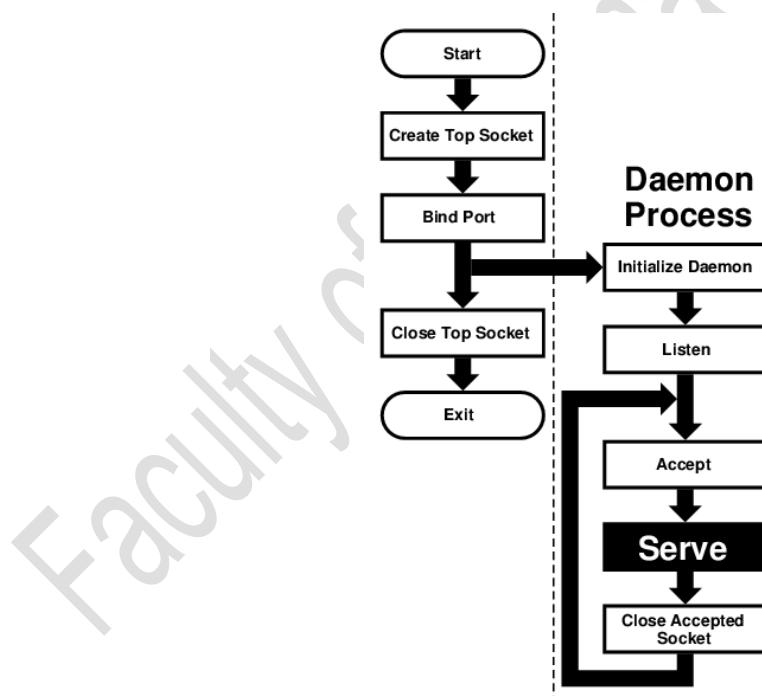
```

ข้อผิดพลาดที่เกิดขึ้นเนื่องจากโปรแกรมฝั่งเซิร์ฟเวอร์ยังไม่ได้ทำงานนั้นเอง ดังนั้นผู้เขียนโปรแกรมต้องสั่งรันโปรแกรมเซิร์ฟเวอร์ก่อนในคลอเอนต์เสมอ

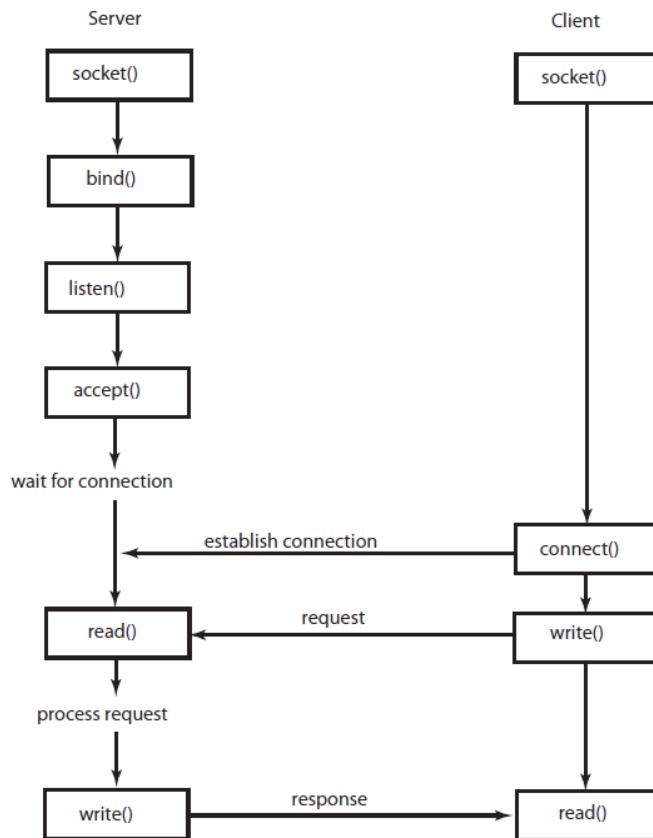
การเขียนโปรแกรมฝั่งเซิร์ฟเวอร์

โปรแกรมในฝั่งเซิร์ฟเวอร์จะแตกต่างจากฝั่งไคลเอนต์เล็กน้อยคือ เมื่อโปรแกรมฝั่งไคลเอนต์เชื่อมต่อกับเซิร์ฟเวอร์แล้ว จากนั้นไคลเอนต์จะรับและส่งข้อมูลกับเซิร์ฟเวอร์อย่างต่อเนื่อง เมื่อรับส่งข้อมูลเสร็จเรียบร้อยแล้ว ไคลเอนต์จะร้องขอเพื่อยุติการเชื่อมต่อจากเซิร์ฟเวอร์ ซึ่งเป็นอันหมดหน้าที่ของไคลเอนต์ แต่สำหรับเซิร์ฟเวอร์แล้วจะไม่มียุติการทำงาน โดยเซิร์ฟเวอร์จะทำงานต่อไปเรื่อยๆ เพื่อรับการเชื่อมต่อจากไคลเอนต์อื่นๆ ต่อไป สำหรับขั้นตอนการเขียนโปรแกรมในฝั่งเซิร์ฟเวอร์ดังนี้

- 1) สร้าง Socket ซึ่งจะทำให้ได้ช้อกเก็ตอ็อปเจกต์ (Socket Object)
- 2) ผูก (Bind) อ็อปเจกต์ของช้อกเก็ตที่ได้จากขั้นตอนที่ 1 เข้ากับหมายเลขไอพีและเดรสน และพอร์ต
- 3) กำหนดจำนวนไคลเอนต์ที่สามารถเข้าใช้งานเซิร์ฟเวอร์ได้พร้อมๆ กัน
- 4) รอรับการเชื่อมต่อ สื่อสารข้อมูล และยุติการเชื่อมต่อจากไคลเอนต์ แสดงดังรูปที่ 15.5



รูปที่ 15.5 (ก) แสดงการทำงานของเซิร์ฟเวอร์



รูปที่ 15.5 (ข) แสดงภาพรวมการทำงานของไคลเอนต์ - เชิร์ฟเวอร์

สำหรับตัวอย่างโปรแกรมฝึกเชิร์ฟเวอร์แสดงได้ดังนี้

Program Example 15.6: Easy Server Script (send day/time)

```

1  ''' Server for providing the day and time.'''
2  from socket import *
3  from time import ctime
4
5  HOST = 'localhost'
6  PORT = 5000
7  BUFFER_SIZE = 1024
8  ADDRESS = (HOST, PORT) #(127.0.0.1, 5000)
9
⇒10 server = socket(AF_INET, SOCK_STREAM)
⇒11 server.bind(ADDRESS)
⇒12 server.listen(5)
13
⇒14 while True:
⇒15     print('waiting for connection...')
⇒16     client, address = server.accept()
⇒17     print('connected from: ', address)
⇒18     data_bytes = str.encode(ctime())
⇒19     client.send(data_bytes)
⇒20     client.close()

```

จากตัวอย่างโปรแกรมที่ 15.6 แสดงโปรแกรมผังเซิร์ฟเวอร์ ทำหน้าที่รับการเชื่อมต่อจากไคลเอนต์ และจะส่งวันและเวลาบนเครื่องเซิร์ฟเวอร์ให้กับโปรแกรมผังไคลเอนต์นำไปแสดงผล บรรทัดที่ 10 สร้าง Socket ของเซิร์ฟเวอร์ชื่อ server เมื่อونกับในตัวอย่างที่ 15.5 บรรทัดที่ 11 โปรแกรมผูกไโอบีแอดเดรส และพอร์ตเข้ากับอ็อกป์เจกต์ชื่อ server บรรทัดที่ 12 กำหนดให้เซิร์ฟเวอร์สามารถรับการเชื่อมต่อจากไคลเอนต์ได้พร้อมๆ กันเท่ากับ 5 เครื่อง บรรทัดที่ 14 กำหนดให้โปรแกรมทำงานตลอดเวลาเพื่อรับการเชื่อมต่อจากไคลเอนต์ บรรทัดที่ 15 เซิร์ฟเวอร์สั่งพิมพ์ข้อความว่า 'waiting for connection...' จากนั้นเซิร์ฟเวอร์จะเฝ้ารการเชื่อมต่อจากไคลเอนต์ เมื่อไคลเอนต์เชื่อมต่อเข้ามา (บรรทัดที่ 16) และเซิร์ฟเวอร์ตอบรับการเชื่อมต่อแล้ว เซิร์ฟเวอร์จะพิมพ์ข้อความว่า 'connected from: ' ตามด้วยหมายเลขไโอบีและพอร์ตตามลำดับดังในบรรทัดที่ 17 จากนั้นโปรแกรมจะดึงวันและเวลาจากระบบด้วยเมธอด ctime() จากโมดูล time แต่ในไฟรอนเวอร์ชัน 3.0 ขึ้นไป จะไม่รองรับการส่งสตริงผ่าน Socket จึงจำเป็นต้องเปลี่ยนวันและเวลาเป็นข้อมูลชนิดไบต์เสียก่อนด้วยเมธอด str.encode(ctime()) และเก็บไว้ในตัวแปรชื่อ data_byte แสดงในบรรทัดที่ 18 จากนั้นโปรแกรมจะส่งวันเวลาที่ถูกแปลงเป็นไบต์แล้วให้กับเครื่องไคลเอนต์ด้วยเมธอด send() เมื่อเซิร์ฟเวอร์ส่งข้อมูลให้ไคลเอนต์เสร็จเรียบร้อยแล้ว จะปิดการเชื่อมต่อทันทีด้วยเมธอด close()

วิธีการรันโปรแกรมเซิร์ฟเวอร์และไคลเอนต์

เนื่องจากไฟรอนอนุญาตให้ผู้ใช้ในโปรแกรมสามารถรันโปรแกรมได้ครั้งละ 1 โปรแกรมเท่านั้น จาก Python Shell หรือ Python IDLE ดังนั้นเมื่อจะสั่งรันโปรแกรมไคลเอนต์-เซิร์ฟเวอร์พร้อมๆ กันบน Python IDLE จะทำให้เซิร์ฟเวอร์ปิดพอร์ตและยุติการทำงานทันที เพื่อให้การรันโปรแกรมดังกล่าวสามารถทำงานได้จะต้องใช้ MS-DOS ช่วยดังนี้

การรันโปรแกรมเซิร์ฟเวอร์

- สั่งรันโปรแกรมผังเซิร์ฟเวอร์จาก Python IDLE จะทำให้ได้ผลลัพธ์ดังรูปด้านล่าง

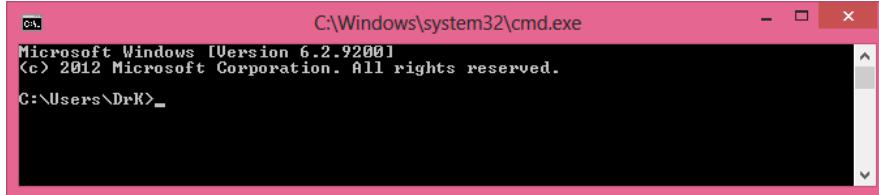
```
>>> ===== RESTART =====
>>>
waiting for connection...
```

- เมื่อสั่งรันโปรแกรมไคลเอนต์จะทำให้เซิร์ฟเวอร์แสดงผลลัพธ์ดังรูปด้านล่าง

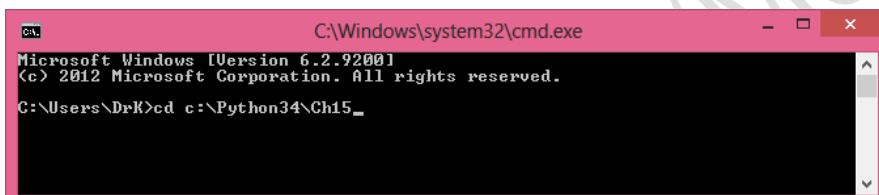
```
>>> ===== RESTART =====
>>>
waiting for connection...
connected from: ('127.0.0.1', 63632)
waiting for connection...
connected from: ('127.0.0.1', 63643)
waiting for connection...
```

การรันโปรแกรมไคลเอนต์

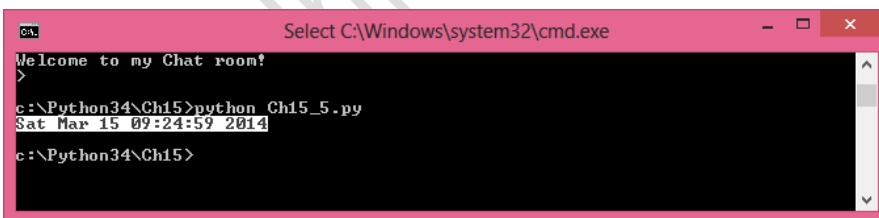
- เลือก start ของวินโดวส์ \Rightarrow Run \Rightarrow พิมพ์คำว่า cmd และกดปุ่ม Enter ดังรูป



- เปลี่ยนไดเรคทรอรี่ไปยังที่เก็บโปรแกรมต้นฉบับของไคลเอนต์ (ในที่นี้ชื่อ Ch15_5.py) จากตัวอย่างนี้โปรแกรมจะเก็บอยู่ใน C:\Python34\Ch15 ดังนั้นให้ใช้คำสั่งเปลี่ยนไดเรคทรอรี่ดังนี้ cd c:\Python34\Ch15 ดังรูป



- เมื่อกดปุ่ม Enter จะเปลี่ยนไดเรคทรอรี่ไปยัง c:\Python34\Ch15 ทันที จากนั้นใช้คำสั่งเพื่อรันโปรแกรมไคลเอนต์ดังนี้ c:\Python34\Ch15>**python** Ch15_5.py ผลลัพธ์แสดงดังรูปด้านล่าง



ไคลเอนต์จะสั่งพิมพ์วันและเวลาที่ส่งมาจากเซิร์ฟเวอร์บันจօภาพ ผู้ใช้งานสามารถสั่งรันโปรแกรมไคลเอนต์ได้เรื่อยๆ ไม่จำกัดจำนวน แต่ไคลเอนต์จะเข้าใช้งานเซิร์ฟเวอร์ได้พร้อมๆ กันได้เพียง 5 เครื่องเท่านั้น

4. การเขียนโปรแกรมพูดคุยผ่านเน็ตเวิร์ค (Chat)

จากตัวอย่างโปรแกรมในหัวข้อที่ผ่านมาเป็นการสื่อสารในลักษณะทางเดียว คือไคลเอนต์รับข้อมูลจากเซิร์ฟเวอร์อย่างเดียวโดยที่ไคลเอนต์ไม่ได้ส่งข้อมูลใดๆ กลับไปให้กับเซิร์ฟเวอร์เลย ดังนั้นในตัวอย่างต่อไปนี้ ผู้เขียนจะแนะนำการเขียนโปรแกรม Chat ซึ่งมีความสามารถในการสื่อสารแบบสองทิศทาง คือผู้ใช้งานสามารถพิมพ์ข้อความได้ต่อบกับเซิร์ฟเวอร์ได้ตลอดเวลา จนกว่าผู้ใช้งานจะยุติ (ปิดโปรแกรม) การทำงานลง แต่ในโปรแกรมนี้ยังมีข้อจำกัดคือ โปรแกรมสามารถโต้ตอบระหว่าง

เซิร์ฟเวอร์และไคลเอ็นต์ในขณะนี้ๆ ได้เพียง 1 ต่อ 1 เท่านั้น สำหรับไคลเอ็นต์อื่นๆ ที่ต้องการสื่อสารกับเซิร์ฟเวอร์จะต้องรอให้ไคลเอ็นต์ที่กำลังทำงานอยู่ติดการทำงานเสียก่อน จึงจะสามารถสื่อสารกับเซิร์ฟเวอร์ดังกล่าวได้ สำหรับการแก้ปัญหาเกี่ยวกับข้อจำกัดของการสื่อสารที่กล่าวมาแล้วจะแสดงในหัวข้อถัดไป

ตัวอย่างโปรแกรม Chat ในฝั่งเซิร์ฟเวอร์แสดงได้ดังนี้

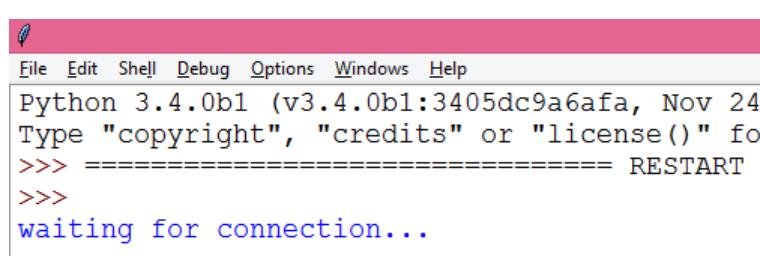
Program Example 15.7: Chat Server Script

```

1  '''Chat Server Script'''
2  from socket import *
3
4  HOST = 'localhost'
5  PORT = 5000
6  BUFFER_SIZE = 1024
7  ADDRESS = (HOST, PORT) # (127.0.0.1, 5000)
8
9  server = socket(AF_INET, SOCK_STREAM)
10 server.bind(ADDRESS)
11 server.listen(5)
12
⇒13 while True:
⇒14     print('waiting for connection...')
⇒15     client, address = server.accept()
⇒16     print('connected from: ', address)
⇒17     client.send(str.encode('Welcome to my Chat room!'))
18
⇒19     while True:
⇒20         message = bytes.decode(client.recv(BUFFER_SIZE))
⇒21         if not message:
⇒22             print("Client disconnected")
⇒23             client.close()
⇒24             break
⇒25         else:
⇒26             print(message)
⇒27             client.send(str.encode(input('> ')))

```

เมื่อสั่งรันโปรแกรม Chat เซิร์ฟเวอร์จะได้ผลลัพธ์ดังนี้



จากตัวอย่างโปรแกรมที่ 15.7 แสดงโปรแกรม Chat เซิร์ฟเวอร์ ทำหน้าที่รับการเชื่อมต่อ (บรรทัดที่ 14, 15) จากไคลเอ็นต์ เมื่อไคลเอ็นต์เชื่อมต่อเข้ามาและเซิร์ฟเวอร์ยอมรับการเชื่อมต่อแล้ว เซิร์ฟเวอร์จะพิมพ์ข้อความว่า 'connected from: ' ตามด้วยหมายเลขไอพีและเครื่องไคลเอ็นต์ (บรรทัดที่

16) ต่อจากนั้นเซิร์ฟเวอร์จะส่งข้อความไปให้กับไคลเอนต์คือ "Welcome to my Chat room!" โดยข้อความดังกล่าวจะต้องถูกแปลงเป็นไบต์เสียก่อน (บรรทัดที่ 17) ด้วยเมธอด str.encode() บรรทัดที่ 19 โปรแกรมจะวนลูปเพื่อให้เซิร์ฟเวอร์ทำงานไปเรื่อยๆ แบบไม่รู้จบ เมื่อเซิร์ฟเวอร์ส่งข้อความว่า "Welcome to my Chat room!" ไปให้กับไคลเอนต์แล้ว เซิร์ฟเวอร์จะรอรับข้อมูลจาก Socket ว่าเครื่องไคลเอนต์ส่งข้อมูลมาให้หรือยัง (บรรทัดที่ 20) ถ้ามีการส่งข้อมูลมายังเซิร์ฟเวอร์ ข้อมูลเหล่านั้นจะถูกแปลงจากไบต์เป็นสตริงเสียก่อนด้วยเมธอด bytes.decode() แต่ถ้าไคลเอนต์ไม่ส่งข้อมูลเข้ามา เซิร์ฟเวอร์จะรอไปเรื่อยๆ ข้อมูลที่ถูกแปลงเป็นสตริงแล้วจะนำไปตรวจสอบว่าข้อมูลดังกล่าวมีความยาวหรือไม่ (บรรทัดที่ 21) ถ้าค่าในตัวแปร message เท่ากับ 0 แสดงว่าไคลเอนต์กดปุ่ม Enter เท่านั้น โดยไม่พิมพ์ข้อความใดๆ เลย ส่งผลให้เซิร์ฟเวอร์พิมพ์ข้อความว่า "Client disconnected" และปิดการเชื่อมต่อกับไคลเอนต์นั้นๆ ทันที และหลุดออกจากลูป while ชั้นใน (บรรทัดที่ 19) ทันทีด้วยคำสั่ง break

แต่ถ้าค่าในตัวแปร message ไม่เท่ากับ 0 แสดงว่าไคลเอนต์ส่งข้อมูลเข้ามา เซิร์ฟเวอร์จะพิมพ์ข้อความดังกล่าว (บรรทัดที่ 26) ออกทางจอภาพของผู้ใช้เซิร์ฟเวอร์ และส่งสตริงคำว่า input('> ') ให้กับไคลเอนต์ (บรรทัดที่ 27) เมื่อคำสั่งนี้ไปปรากฏในผู้ใช้ไคลเอนต์จะถูกแปลงความหมายคือ พังชัน input() ซึ่งจะกลายเป็น prompt บนผู้ใช้ไคลเอนต์นั่นเอง สรุปการทำงานของเซิร์ฟเวอร์คือ เซิร์ฟเวอร์จะปิดการเชื่อมต่อจากไคลเอนต์เมื่อไคลเอนต์กดปุ่ม Enter อย่างเดียวเท่านั้น

ตัวอย่างโปรแกรม Chat ในผู้ใช้ไคลเอนต์แสดงได้ดังนี้

Program Example 15.8: Chat Client Script

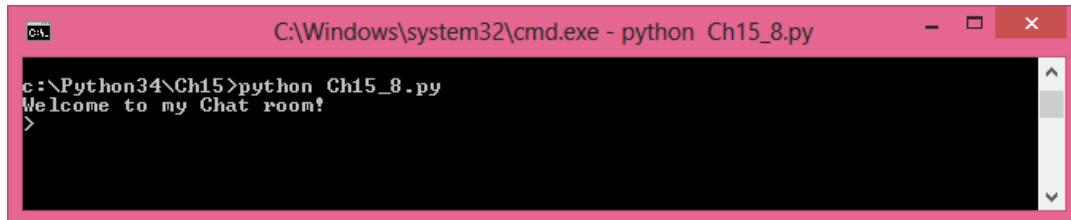
```

1  '''Chat Client Script'''
2  from socket import *
3
4  HOST = 'localhost'
5  PORT = 5000
6  BUFFER_SIZE = 1024
7  ADDRESS = (HOST, PORT) #(127.0.0.1, 5000)
8
9  server = socket(AF_INET, SOCK_STREAM)
10 server.connect(ADDRESS)
11 messageFromServer = bytes.decode(server.recv(BUFFER_SIZE))
12 print(messageFromServer)
13
⇒14 while True:
⇒15     message = input('> ')
⇒16     if not message:
⇒17         break
⇒18     server.send(str.encode(message))
⇒19     reply = bytes.decode(server.recv(BUFFER_SIZE))
⇒20     if not reply:
⇒21         print('Server disconnected')
⇒22         break
⇒23     print(reply)

```

⇒24 | `server.close()`

เมื่อสั่งรันโปรแกรม Chat โคลอีนต์บน MS-DOS จะได้ผลลัพธ์ดังนี้



```
c:\Python34\Ch15>python Ch15_8.py
Welcome to my Chat room!
>
```

ทดสอบพิมพ์ข้อความบนฝั่งโคลอีนต์ เช่น 'I'm client' จะได้ผลลัพธ์ดังนี้

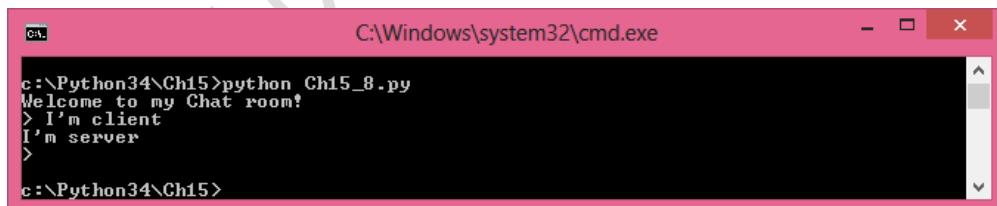


```
c:\Python34\Ch15>python Ch15_8.py
Welcome to my Chat room!
> I'm client
-
```

ข้อความ 'I'm client' จะไปปรากฏที่ฝั่งเซิร์ฟเวอร์ ให้ทดสอบพิมพ์ข้อความว่า 'I'm server' บนฝั่งเซิร์ฟเวอร์

```
>>>
waiting for connection...
connected from: ('127.0.0.1', 51588)
I'm client
> I'm server
```

ข้อความจะปรากฏบนฝั่งโคลอีนต์ดังรูปด้านล่าง หากใช้กดปุ่ม Enter จะทำให้โคลอีนต์ยุติการทำงานทันทีดังรูป



```
c:\Python34\Ch15>python Ch15_8.py
Welcome to my Chat room!
> I'm client
I'm server
>
```

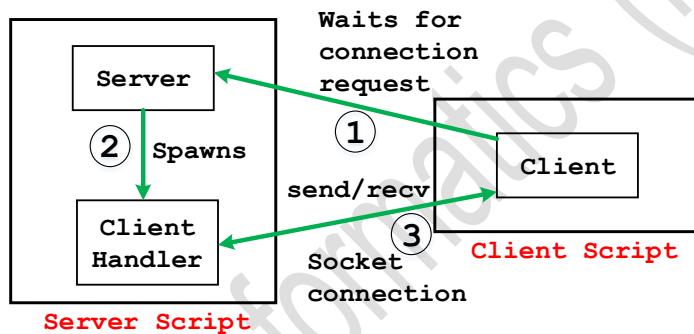
สำหรับโปรแกรมบนฝั่งเซิร์ฟเวอร์จะปิดการเชื่อมต่อของโคลอีนต์ และรอการเชื่อมต่อจากโคลอีนต์เครื่องอื่นๆ ต่อไป

```
>>>
waiting for connection...
connected from: ('127.0.0.1', 51588)
I'm client
> I'm server
Client disconnected
waiting for connection...
```

สำหรับตัวอย่างโปรแกรม Chat Client จะมีลักษณะการทำงานคล้ายกับผู้ใช้เซิร์ฟเวอร์ แต่แตกต่างกันคือ โปรแกรมผู้ใช้คลื่นที่จะบุตการทำงานทันทีเมื่อกดปุ่ม Enter แต่เซิร์ฟเวอร์จะทำงานต่อไปเรื่อยๆ โดยไม่บุตการทำงานจนกว่าผู้ใช้จะปิดโปรแกรม

5. โปรแกรม Chat ที่สามารถรองรับคลื่นพร้อมๆ กันหลายเครื่อง

จากตัวอย่าง Chat ที่ผ่านมาในหัวข้อที่แล้ว มีข้อจำกัดคือ ในขณะหนึ่งๆ จะมีเพียงคลื่นเดียวเท่านั้นที่สามารถสื่อสารกับเซิร์ฟเวอร์ได้ โดยคลื่นอื่นๆ ต้องรอจนกว่าคลื่นอื่นที่กำลังสื่อสารอยู่จะบุตการทำงาน ในตัวอย่างนี้ผู้เขียนจะปรับปรุงให้เซิร์ฟเวอร์สามารถรองรับการเชื่อมต่อจากคลื่นอื่นได้พร้อมๆ กัน โดยใช้คุณสมบัติเรื่องของชีรดในหัวข้อที่ผ่านมาช่วยในการทำงาน เมื่อคลื่นต้องขอรับเชิร์ฟเวอร์ เชิร์ฟเวอร์จะสร้างชีรด (Spawns) เพื่อจัดการสื่อสารกับคลื่นอื่น (Client Handler) เหล่านั้นโดยแยกเป็นอิสระจากกัน แสดงดังรูปที่ 15.6



รูปที่ 15.6 แสดงการทำงานของเซิร์ฟเวอร์ที่สามารถรองรับคลื่นอื่นได้พร้อมๆ กัน

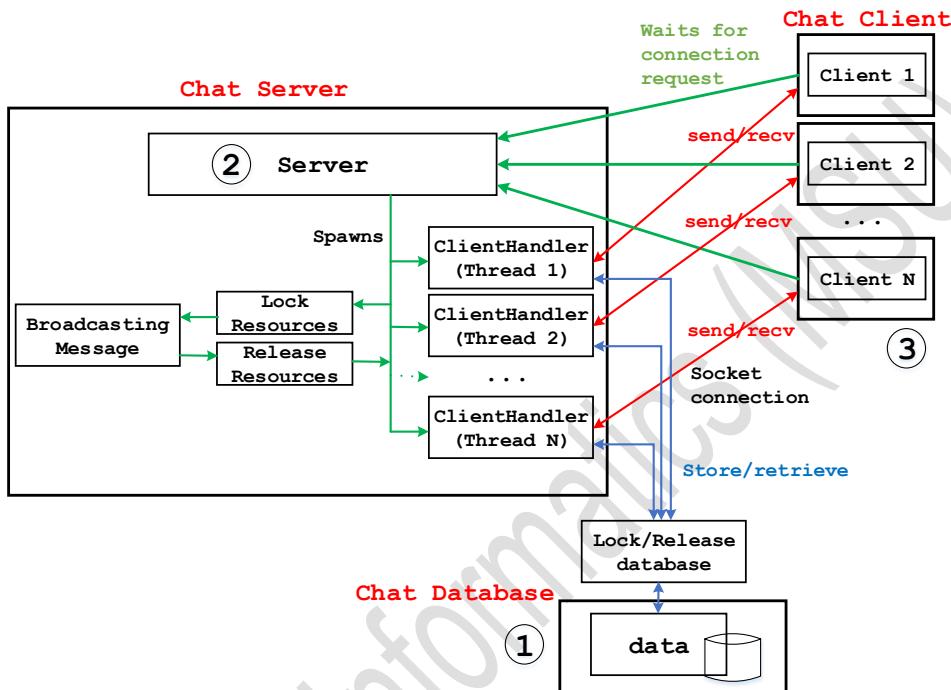
- ❶ คลื่นต้องขอการเชื่อมต่อไปยังเซิร์ฟเวอร์ผ่าน Socket
- ❷ เมื่อเซิร์ฟเวอร์ยอมรับการเชื่อมต่อจากคลื่นอื่นแล้ว เซิร์ฟเวอร์จะสร้างชีรด (Spawns) เพื่อจัดการกับคลื่นอื่น (Client Handler) แต่ละเครื่อง

- ❸ คลื่นอื่นส่งและรับ (send/recv) ข้อมูลกับเซิร์ฟเวอร์ ผ่าน Socket (Socket Connection) บุตการเชื่อมต่อโดยกดปุ่ม Enter โดยปราศจากข้อความใดๆ

ขั้นตอนออกแบบโปรแกรม Chat

1. ออกแบบฐานข้อมูลสำหรับเก็บข้อความที่คลื่นสนทนากัน
2. ออกแบบเซิร์ฟเวอร์ที่สามารถจัดการกับคลื่นอื่นได้หลายเครื่องพร้อมกัน (Spawns) โดยการลงทะเบียนด้วยชื่อผู้ใช้ (User name), เก็บและดึงข้อมูล (Store/Retrieve) การสนทนากลางๆ ให้กับคลื่นอื่น, ส่งข้อความไปยังทุกๆ คลื่น (Broadcasting), บุตการเชื่อมต่อเมื่อผู้ใช้ยกเลิกการใช้งาน (โดยกดปุ่ม Enter ที่

- ปราศจากข้อความใดๆ), ล็อกและปลดปล่อยทรัพยากรที่ใช้ทำงานร่วมกันเพื่อให้ทรัพยากรเหล่านั้นถูกเข้าใช้งานอย่างถูกต้อง (เนื่องจากโปรแกรมเขียนด้วยชีร์ด)
3. ออกแบบไคลเอนต์ที่สามารถสนทนากับไคลเอนต์อื่นๆ ได้พร้อมกันหลายเครื่อง และยุติการทำงานของโปรแกรมด้วยการกดปุ่ม Enter โดยปราศจากข้อความใดๆ สำหรับรูปแบบการทำงานทั้งระบบแสดงในรูปที่ 15.7



รูปที่ 15.7 แสดงภาพรวมของโปรแกรม Chat

สร้างโปรแกรมชื่อ `chatDatabase.py` เพื่อใช้สำหรับเก็บข้อมูลการสนทนา

Program Example `chatDatabase.py`:

```

1  '''Chat Database Script'''
2  class chatRecord():
3      def __init__(self):
4          self.data = []
5
6      def addMessage(self, message):
7          self.data.append(message)
8
9      def getMessage(self, messageID):
10         if len(self.data) == 0:
11             return 'No message yet!'
12         elif messageID == 0: #get all message for database
13             return '\n'.join(self.data)
14         elif messageID != 0: #get a chunk of message
15             temp = self.data[messageID:]

```

```

16             return '\n'.join(temp)
⇒17     else:
18         return "\n"

```

จากตัวอย่างโปรแกรม chatDatabase.py แสดงการสร้างฐานข้อมูลอย่างง่ายเพื่อใช้สำหรับเก็บข้อมูลการสนทนาระหว่างไคลเอนต์ โดยฐานข้อมูลดังกล่าวยังมีจุดอ่อนคือ ข้อมูลการสนทนาจะสูญหายเมื่อยกติการทำงานของเซิร์ฟเวอร์ ถ้าผู้เขียนโปรแกรมต้องการให้ข้อมูลการสนทนาถูกเก็บแบบถาวร ผู้เขียนแนะนำให้เก็บเป็น Text ไฟล์ (หมายความว่าไคลเอนต์ที่มีจำนวนน้อย), เก็บใน XML ไฟล์ (จำนวนไคลเอนต์ปานกลาง) และเก็บในฐานข้อมูล เช่น MySQL, Oracle, Informix (สำหรับไคลเอนต์จำนวนมาก)

บรรทัดที่ 2 - 4 สร้างคลาสชื่อ chatRecord() โดยมีคอนสตรัคเตอร์กำหนดค่าเริ่มต้นให้กับตัวแปรชนิดลิสต์ชื่อ data เพื่อทำหน้าที่เก็บข้อมูลการสนทนาของไคลเอนต์ทั้งหมด บรรทัดที่ 6 – 7 สร้างเมธอดชื่อ addMessage() ทำหน้าที่เพิ่มข้อมูลการสนทนาไว้ในตัวแปร data โดยใช้เมธอด append() มีพารามิเตอร์ 1 ตัวคือ message (ข้อมูลการสนทนา) บรรทัดที่ 9 – 18 สร้างเมธอดชื่อว่า getMessage() ทำหน้าดึงข้อมูลจากฐานข้อมูล (ในตัวแปร data) โดยมีพารามิเตอร์ 1 ตัว (messageID) คือ หมายเลขบรรทัดของข้อมูลในฐานข้อมูล ถ้า messageID เท่ากับ 0 โปรแกรมจะดึงข้อมูลทั้งหมดในฐานข้อมูลส่งกลับไปให้กับไคลเอนต์ ถ้า messageID เป็นเลขจำนวนเต็มที่ไม่เท่ากับ 0 โปรแกรมจะดึงข้อมูลตั้งแต่ตำแหน่งบรรทัดใน messageID ถึง ตำแหน่งข้อมูลบรรทัดสุดท้ายของตัวแปร data สำหรับคำสั่ง '\n'.join(self.data) ทำหน้าที่เชื่อมต่อข้อมูลในตัวแปร data เป็นไวยางกัน เช่น data[0] = 'Hello', data[1] = 'World!' และ data[2] = 'Python' เมื่อใช้คำสั่ง '\n'.join(self.data) จะได้ผลลัพธ์คือ 'Hello\nWorld!\nPython' ถ้าตัวแปร data ไม่มีข้อมูลใดๆ เก็บอยู่โปรแกรมจะคืนค่าให้กับไคลเอนต์เป็น 'No message yet!'

สร้างโปรแกรมชื่อ ChatServer.py เพื่อทำหน้าที่เป็นเซิร์ฟเวอร์สำหรับโปรแกรม Chat

Program Example ChatServer.py

```

1      '''Chat Server for a multi-client chat room'''
2      from socket import *
⇒3      from chatDatabase import chatRecord
⇒4      from threading import Thread
5      import threading
6      from time import ctime
7
8      class clientHandler(Thread):
9          def __init__(self, client, record, address):
10             Thread.__init__(self)
11             self._client = client
12             self._record = record
13             self._address = address
14
15             #broadcasting chat messages to all connected clients

```

```

⇒16    def broadCastingMessage(self, activeClient, message):
17        #Do not send the message to server and the client who has
18        send the message to us
⇒19            for socket in CONNECTIONS_LIST:
20                if socket != server and socket != activeClient:
21                    try:
22                        broadcastMessage = str.encode(message)
23                        socket.send(broadcastMessage)
24                    except:
25                        # broken socket connection may be, chat
26                        client pressed ctrl+c for example
27                        print ("Client (%s) is offline"
28                               %self._address)
29                        broadCastingMessage(socket, ("Client (%s)
is offline" %self._address))
30                        socket.close()
31                        CONNECTIONS_LIST.remove(socket)
32
33            def run(self):
34                self._client.send(str.encode('Welcome to the chat
room'))
35                self._name = bytes.decode(self._client.recv(BUFSIZE))
36                #Geting all messages from database and send them to
client in the first time
37                allMessage = self._record.getMessage(0)
38                self._client.send(str.encode(allMessage))
39                while True:
40                    message =
41                    bytes.decode(self._client.recv(BUFSIZE))
42                    if not message:
43                        print('Client disconnected')
44                        self._client.close()
45                        CONNECTIONS_LIST.remove(self._client)
46                        break
47                    else:
48                        message = ctime() + ': [' + self._name + '] -
->' + message
49                        self._record.addMessage(message)
50                        #Broadcasting a new messages to every clients
51                        threadLock.acquire()
52                        self.broadCastingMessage(self._client,
53                                     message)
54                        threadLock.release()
55
56 HOST = 'localhost'
57 PORT = 5000
58 BUFSIZE = 4096
59 ADDRESS = (HOST, PORT) #(127.0.0.1, 5000)
# List to keep track of all socket connections
CONNECTIONS_LIST = []
# Creating Threads Synchronization
threadLock = threading.Lock()
record = chatRecord()

```

```

60     server = socket(AF_INET, SOCK_STREAM)
61     server.bind(ADDRESS)
62     server.listen(10)
63     # Add server socket to the list
64     CONNECTIONS_LIST.append(server)
65     print ("Chat server started on port " + str(PORT))
66
67     while True:
68         print('Waiting for connection...')
69         client, address = server.accept()
70         print('...connected from:', address)
71         # Lock CONNECTIONS_LIST for inserting connected client
72         threadLock.acquire()
73         CONNECTIONS_LIST.append(client)
74         # Release CONNECTIONS_LIST
75         threadLock.release()
76         handler = clientHandler(client, record, address)
77         handler.start()

```

จากตัวอย่างโปรแกรม ChatServer.py แสดงการสร้างเซิร์ฟเวอร์ Chat บรรทัดที่ 3 นำเข้าฐานข้อมูลที่ได้สร้างไว้แล้วคือ แฟ้ม chatDatabase.py เข้ามาทำงาน โดยใช้คำสั่ง form chatDatabase import chatRecord (ชื่อคลาสในแฟ้ม chatDatabase.py) บรรทัดที่ 4 – 5 นำเข้าโมดูล threading เพื่อใช้สำหรับสร้าง-thread ของ ไคลเอนต์แต่ละเครื่อง บรรทัดที่ 6 นำเข้าโมดูล time เพื่อใช้สร้างวันและเวลา

บรรทัดที่ 8 สร้างคลาสชื่อ clientHandler() ทำหน้าที่จัดการไคลเอนต์ที่เชื่อมต่อเข้ามายังเซิร์ฟเวอร์ Chat โดยคลาสดังกล่าวสืบทอดคุณสมบัติมาจากคลาส Thread ซึ่งส่งผลให้คลาส clientHandler มีคุณสมบัติเป็นชุดไปด้วย ในคลาสนี้สร้างคอนสตรักเตอร์ (บรรทัดที่ 9 - 13) ทำหน้าที่กำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ ก่อนคลาสดังกล่าวจะทำงาน สำหรับค่าเริ่มต้นที่กำหนดคือ ออปเจ็กต์ของไคลเอนต์ (self._client), ตัวแปรที่อ้างอิงไปยังฐานข้อมูล (self._record) และหมายเลขไอพี แอดเดรส (self._address) บรรทัดที่ 16 โปรแกรมสร้างเมธอดชื่อ broadCastingMessage() ทำหน้าที่ส่งข้อความสนทนาจากไคลเอนต์เครื่องใดเครื่องหนึ่งไปยังทุกๆ เครื่องที่กำลังสื่อสารกับเซิร์ฟเวอร์ Chat อยู่ เมธอดนี้ต้องการพารามิเตอร์ 2 ตัวคือ ออปเจ็กต์ (activeClient) ของไคลเอนต์เครื่องใดเครื่องหนึ่งที่กำลังเชื่อมต่ออยู่ และข้อความ (message) ที่ไคลเอนต์ต้องการส่งไปยังไคลเอนต์อื่นๆ ทั้งหมดที่อยู่ในระบบ บรรทัดที่ 18 โปรแกรมนำเอาอ้อปเจ็กต์ของ Socket ที่เก็บรวบรวมไว้ในตัวแปร CONNECTIONS_LIST มาตรวจสอบว่าเป็นอ้อปเจ็กต์ Socket ของเซิร์ฟเวอร์หรือไม่ หรือเป็นอ้อปเจ็กต์ของตนเองหรือไม่ อธิบายง่ายๆ คือ ถ้าไคลเอนต์ A ส่งข้อความไปยังไคลเอนต์อื่นๆ ในระบบ ข้อความดังกล่าวไม่ควรถูกส่งไปยัง Socket ของเซิร์ฟเวอร์และ Socket ของตัวเอง (ไคลเอนต์ A) โดยใช้เงื่อนไขในโปรแกรมบรรทัดที่ 19 ต่อจากนั้นบรรทัดที่ 21 โปรแกรมวนลูปส่งข้อความสนทนาไปยังทุกๆ เครื่องยกเว้นเซิร์ฟเวอร์และตัวเอง ถ้าโปรแกรมไม่สามารถส่งข้อความไปยังเครื่องไคลเอนต์ใดๆ ได้ เซิร์ฟเวอร์จะส่งข้อความไปบอกไคลเอนต์อื่นๆ ว่ามีไคลเอนต์บางตัวไม่สามารถเชื่อมต่อได้ และ

โปรแกรมจะทำการปิด Socket ของไคลเอนต์เครื่องที่ไม่ตอบสนอง และลบอีก็อปเจกต์ของไคลเอนต์ ดังกล่าวออกจากตัวแปร CONNECTIONS_LIST (บรรทัดที่ 25 - 28) โดยคำสั่งทั้งหมดจะทำงานอยู่ภายใต้การควบคุมของคำสั่ง try...except

บรรทัดที่ 30 โปรแกรมทำการໂອເວັຣ໌ເມນີໂຫຼດ run() ຂອງຄລາສ Thread ເພື່ອທຳນັກທີ່ຄວບຄຸມ
ການສະຖານະຫວ່າງໄຄລເອັນຕ ໂດຍເຮີ່ມຕົ້ນເຊື່ອຟເວັຣ໌ຈະສັ່ງຂ້ອງຄວາມວ່າ 'Welcome to the chat room'
(บรรทัดທີ່ 31) ໄປໃຫ້ໄຄລເອັນຕກ່ອນ ຈາກນັ້ນໄຄລເອັນຕຈະສັ່ງຊື່ຜູ້ໃຊ້ກລັບໄປໃຫ້ເຊື່ອຟເວັຣ໌ (บรรทัดທີ່ 32)
ຂັ້ນຕອນຕ່ອງໄປເຊື່ອຟເວັຣ໌ຈະສັ່ງຂ້ອມລາກສະຖານະທີ່ມີຢູ່ທັງໝົດໃນຮູ້ານຂ້ອມລີໄປໃຫ້ກັບໄຄລເອັນຕ (บรรทัดທີ່
34 - 35) ຈາກນັ້ນເຊື່ອຟເວັຣ໌ຈະທຳການວຸ່ນລູ່ປັດວ່າຍຄໍາສັ່ງ while True: ຜຶ່ງຈະທຳໃຫ້ໂປຣແກຣມທຳການໄປເຮືອງຖ້າ
ຈົນກວ່າຈະປິດໂປຣແກຣມ ບຣທັດທີ່ 37 – 49 ໂປຣແກຣມເຊື່ອຟເວັຣ໌ຈະຮັບຂ້ອມລາກໄຄລເອັນຕເຄື່ອງໄດ້ເຄື່ອງ
ໜຶ່ງທີ່ສັ່ງເຂົ້າມາ ແລະສັ່ງຂ້ອງຄວາມດັກລ່າໄປໃຫ້ກັບໄຄລເອັນຕເຄື່ອງອື່ນໆ ພຣ້ອມກັນທັງໝົດ ຄ້າໄຄລເອັນຕ
ເຄື່ອງໄດ້ ກດປຸ່ມ Enter 1 ຄຽ້ງ ເຊື່ອຟເວັຣ໌ຈະຄື່ອວ່າເປັນກາຍຸດືການເຊື່ອມຕ່ອງຂອງໄຄລເອັນຕເຄື່ອງນັ້ນໆ
(บรรທັດທີ່ 38) ແຕ່ຄ້າເປັນຂ້ອງຄວາມປົກຕ ເຊື່ອຟເວັຣ໌ຈະເຮັກໃຊ້ງານເມນີໂຫຼດ broadCastingMessage() ເພື່ອ
ສັ່ງຂ້ອງຄວາມໄປຢັງທຸກໆ ໄຄລເອັນຕ ແຕ່ການເຮັກໃຊ້ມີໂຫຼດ broadCastingMessage() ອາຈະຖຸກເຮັກໃຊ້ຈາກ
ເຫຼືດອື່ນໆ ພຣ້ອມໆ ກັນໄດ້ ດັ່ງນັ້ນເຊື່ອຟເວັຣ໌ຈະໃຊ້ຄຸນສມບັດກາລືອດ (threadLock.acquire()) ເມນີໂຫຼດ
broadCastingMessage() ໄວກ່ອນ ເພື່ອໄມ່ໃຫ້ເຫຼືດອື່ນໆ ເຂົ້າມາແຍ່ງໃຊ້ງານ ເມື່ອໃຊ້ງານເມນີໂຫຼດ
broadCastingMessage() ເສົ່ງຈະເຮັກໃຊ້ມີໂຫຼດ ເຊື່ອຟເວັຣ໌ຈະປັດປຸງທັງໝົດ
threadLock.release() ເພື່ອໃຫ້ເຫຼືດ (ໄຄລເອັນຕເຄື່ອງອື່ນໆ) ອື່ນໆ ເຮັກໃຊ້ມີໂຫຼດ
broadCastingMessage() ບ້າງ

บรรทัดที่ 53 ผู้เขียนทำการเปลี่ยนขนาดของบัฟเฟอร์จาก 1024 จำนวนตัวอย่างที่ผ่านมาเป็น 4098 เนื่องจากต้องการให้โปรแกรมสามารถรับข้อมูลที่มีขนาดยาวขึ้น บรรทัดที่ 56 สร้างตัวแปรชนิดลิสต์ชื่อ CONNECTIONS_LIST เพื่อใช้สำหรับเก็บอ้อปเจกต์ของ Socket คลอเอนต์ทุกๆ เครื่อง เอาไว้ รวมถึงอ้อปเจกต์ Socket ของเซิร์ฟเวอร์ด้วย เพราะต้องการใช้อ้างอิงในกรณีที่ต้องการส่ง ข้อมูลไปยังทุกๆ เครื่อง คลอเอนต์นั้นเอง บรรทัดที่ 58 สร้างตัวแปรสำหรับใช้ล็อกและปลดปล่อย ทรัพยากรในกรณีที่อาจจะเกิดการແย่งชิงกันเข้าใช้งานของ คลอเอนต์ชุด บรรทัดที่ 59 สร้างอินสแตนซ์ ของคลาส chatDatabase เพื่อใช้เก็บข้อมูลการสนทนา บรรทัดที่ 64 เพิ่มอ้อปเจกต์ Socket ของ เซิร์ฟเวอร์ไว้ในตัวแปร CONNECTIONS_LIST เพื่อใช้สำหรับอ้างอิงในอนาคต บรรทัดที่ 67 - 77 เซิร์ฟเวอร์ทำการวนลูปอัปเดตการเชื่อมต่อจาก คลอเอนต์ เมื่อมี คลอเอนต์ร้องขอเข้ามา และเซิร์ฟเวอร์ ยอมรับการเชื่อมต่อดังกล่าวแล้ว เซิร์ฟเวอร์จะเก็บอ้อปเจกต์ Socket ของ คลอเอนต์ไว้ในตัวแปร CONNECTIONS_LIST ไปเรื่อยๆ จากนั้นเซิร์ฟเวอร์จะสร้างชีร์ดของ คลอเอนต์ โดยมีพารามิเตอร์ 3 ตัวคือ อ้อปเจกต์ Socket ของ คลอเอนต์ (client), อ้อปเจกต์ฐานข้อมูล (record) และหมายเลขไอพี

แอดเดรส (address) แล้วเซิร์ฟเวอร์ก็ทำการเริ่มต้น herdทันที (บรรทัดที่ 77) ส่งผลให้ herdแต่ละ herdที่สร้างขึ้นจะดูแลโคลอินต์แต่ละเครื่องแบบ 1 herd ต่อ 1 เครื่องนั่นเอง



Note: ในตัวอย่างที่ผ่านมาทั้งหมด โปรแกรมจะทำงานอยู่ภายใต้เครื่องเดียวเท่านั้น ถ้าผู้เขียนโปรแกรมต้องการให้เซิร์ฟเวอร์และโคลอินต์ทำงานอยู่ต่างเครื่องกัน ให้กำหนด HOST จาก 'localhost' เป็นหมายเลขไอพีแอดเดรสแทน เช่น HOST = '192.168.1.10' และโคลอินต์จะต้องกำหนด HOST ให้เหมือนกับผู้เซิร์ฟเวอร์ด้วย

สร้างโปรแกรมชื่อ ChatClient.py เพื่อทำหน้าที่เป็นโคลอินต์

Program Example ChatClient.py

```

1  '''Chat Client for a multi-client chat room'''
2  from socket import *
3
4  HOST = 'localhost'
5  PORT = 5000
6  BUFSIZE = 4096
7  ADDRESS = (HOST, PORT) #(127.0.0.1, 5000)
8
9  server = socket(AF_INET, SOCK_STREAM)
10 server.connect(ADDRESS)
11 messageFromServer = bytes.decode(server.recv(BUFSIZE))
12 print(messageFromServer)
13 name = input('Enter your name: ')
14 userName = str.encode(name)
15 server.send(userName)
16
17 while True:
18     receiveMessage = bytes.decode(server.recv(BUFSIZE))
19     if not receiveMessage:
20         print('Server disconnected')
21         break
22     print(receiveMessage)
23     sendMessage = input('> ')
24     if not sendMessage:
25         print('Server disconnected')
26         break
27     server.send(str.encode(sendMessage))
28 server.close()

```

จากตัวอย่างโปรแกรม ChatClient.py แสดงการสร้างโคลอินต์ Chat ซึ่งเหมือนในตัวอย่างที่ 15.8 ทุกประการ แตกต่างกันคือ ขนาดของบัฟเฟอร์กำหนดให้มีขนาดใหญ่ขึ้นจาก 1024 เป็น 4096 ดังในบรรทัดที่ 6

ขั้นตอนการสั่งรับโปรแกรม ChatServer และ ChatClient

- 1) สั่งรัน ChatServer.py ผ่านทาง Python IDLE โดยการกดปุ่ม F5 หรือเลือกเมนู Run  Run Module F5 ดังรูป

```
>>> ===== RESTART =====
>>>
Chat server started on port 5000
Waiting for connection...
```

- 2) สั่งรัน ChatClient.py ผ่านทาง MS-DOS โดยใช้คำสั่ง python ChatClient.py และกดปุ่ม Enter จากนั้นให้ทดสอบ Chat ดังรูปด้านล่าง



Three terminal windows showing the interaction between ChatServer.py and multiple ChatClient.py instances.

- Terminal 1 (Left):** Shows the ChatServer.py process running on port 5000, waiting for connections.
- Terminal 2 (Middle):** Shows a client named BBB connecting and having a conversation with another client (John).
- Terminal 3 (Right):** Shows a client named AAA connecting and having a conversation with another client (Manee).

```
C:\Windows\system32\cmd.exe - python Client.py
c:\Python34\Ch15>python Client.py
Welcome to the chat room
Enter your name: BBB
No message yet!
> I am B
Sun Mar 16 14:32:02 2014: [AAA] -->I am A
> Hi every one
Sun Mar 16 14:32:33 2014: [AAA] -->Hello
> To day is so good
Sun Mar 16 14:33:15 2014: [AAA] -->Why do you feel good?
>
Server disconnected

c:\Python34\Ch15>python Client.py
Welcome to the chat room
Enter your name: John
No message yet!
> I'm John
Sun Mar 16 18:54:59 2014: [Manee] -->I'm Manee
> Hello
Sun Mar 16 18:55:07 2014: [Somsri] -->I'm Somsri
> I'm playing computer
Sun Mar 16 18:55:27 2014: [Manee] -->How do you do?Sun Mar 16 18:55:36 2014: [Somsri] -->What are you doing?Sun Mar 16 18:55:51 2014: [John] -->I'm playing computer
> I'm chatting now

...connected from: ('127.0.0.1', 62340)
Waiting for connection...
Client disconnected
Client disconnected
Client disconnected

>>> ===== RESTART =====
>>>
Chat server started on port 5000
Waiting for connection...
...connected from: ('127.0.0.1', 64354)
Waiting for connection...
...connected from: ('127.0.0.1', 64358)
Waiting for connection...
...connected from: ('127.0.0.1', 64359)
Waiting for connection...

C:\Windows\system32\cmd.exe - python Client.py
c:\Python34\Ch15>python Client.py
Welcome to the chat room
Enter your name: AAA
No message yet!
> I am A
Sun Mar 16 14:32:02 2014: [BBB] -->I am B
Sun Mar 16 14:32:02 2014: [BBB] -->Hi every one
> To day is so good
Sun Mar 16 14:32:33 2014: [AAA] -->Hello
Sun Mar 16 14:32:50 2014: [BBB] -->To day is so good
Sun Mar 16 14:33:15 2014: [AAA] -->Why do you feel good?
>
Server disconnected

c:\Python34\Ch15>python Client.py
Welcome to the chat room
Enter your name: Manee
No message yet!
> I'm Manee
Sun Mar 16 18:54:49 2014: [John] -->I'm John
> How do you do?
> Hello!
> I'm chatting now
Sun Mar 16 18:55:07 2014: [Somsri] -->I'm SomsriSun Mar 16 18:55:17 2014: [John] -->I'm Somsri
> I'm chatting now
Sun Mar 16 18:55:36 2014: [Somsri] -->What are you doing?Sun Mar 16 18:55:51 2014: [John] -->I'm playing computer
> I'm chatting now

...connected from: ('127.0.0.1', 62340)
Waiting for connection...
Client disconnected
Client disconnected
Client disconnected

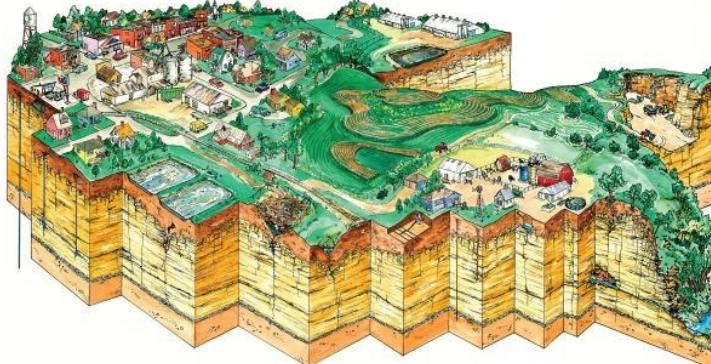
>>> ===== RESTART =====
>>>
Chat server started on port 5000
Waiting for connection...
...connected from: ('127.0.0.1', 64354)
Waiting for connection...
...connected from: ('127.0.0.1', 64358)
Waiting for connection...
...connected from: ('127.0.0.1', 64359)
Waiting for connection...
```

จบบทที่ 15

บทที่ 16

การเขียนโปรแกรมระบบสารสนเทศภูมิศาสตร์เบื้องต้น

(Introduction to Geographic Information System Programming: GISP)



ที่มา: www.winona.edu

1. ระบบสารสนเทศภูมิศาสตร์ (Geographic Information System: GIS)

ระบบสารสนเทศภูมิศาสตร์ คือกระบวนการทำงานเกี่ยวกับข้อมูลในเชิงพื้นที่ด้วยระบบคอมพิวเตอร์ เพื่อใช้กำหนดข้อมูลและสารสนเทศที่มีความสัมพันธ์กับตำแหน่งในเชิงพื้นที่ เช่น ที่อยู่อาศัย ทรัพยากรธรรมชาติ โดยสัมพันธ์กับตำแหน่งในแผนที่ เส้นรุ้ง และเส้นทาง

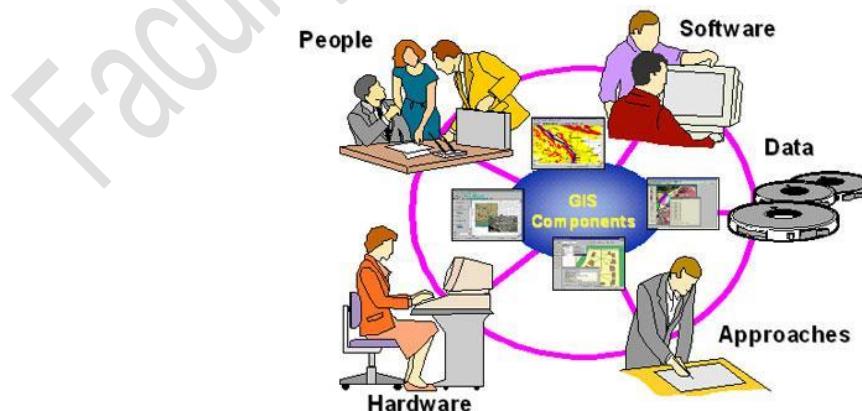
ข้อมูลและแผนที่ใน GIS เป็นระบบข้อมูลสารสนเทศที่อยู่ในรูปของตารางข้อมูลและฐานข้อมูลซึ่งรูปแบบและความสัมพันธ์ของข้อมูลเชิงพื้นที่ทั้งหลายสามารถนำมาระบุตำแหน่งได้โดย GIS เพื่อทำให้สื่อความหมายในเรื่องการเปลี่ยนแปลงที่สัมพันธ์กับเวลาได้ เช่น การแพร่ขยายของโรคระบาด การเคลื่อนย้ายถิ่นฐาน การบุกรุกทำลาย การเปลี่ยนแปลงของการใช้พื้นที่ ฯลฯ ข้อมูลเหล่านี้ เมื่อประยุกต์บนแผนที่ทำให้สามารถแปลงและสื่อความหมายใช้งานได้ง่าย (ข้อมูลตั้งตนบัญชีทั้งหมดคัดลอกมาจาก <http://www.gisthai.org/>)

GIS เป็นระบบข้อมูลข่าวสารที่เก็บไว้ในคอมพิวเตอร์ แต่สามารถแปลงความหมายเชื่อมโยงกับสภาพภูมิศาสตร์อื่นๆ สภาพการทำงานของระบบสัมพันธ์กับสัดส่วนระหว่างและพื้นที่จริงบนแผนที่ ข้อมูลที่จัดเก็บใน GIS มีลักษณะเป็นข้อมูลเชิงพื้นที่ (Spatial Data) ที่แสดงในรูปของภาพ (graphic) แผนที่ (map) ที่เชื่อมโยงกับข้อมูลเชิงบรรยาย (Attribute Data) หรือฐานข้อมูล (Database) การเชื่อมโยงข้อมูลทั้งสองประเภทเข้าด้วยกัน จะทำให้ผู้ใช้สามารถที่จะแสดงข้อมูลทั้งสองประเภทได้พร้อมๆ กัน

องค์ประกอบของ GIS (Components of GIS)

องค์ประกอบหลักของระบบ GIS จัดแบ่งออกเป็น 5 ส่วนใหญ่ๆ คือ อุปกรณ์คอมพิวเตอร์ (Hardware) โปรแกรม (Software) ขั้นตอนการทำงาน (Methods) ข้อมูล (Data) และบุคลากร (People) ดังรูปที่ 16.1 โดยมีรายละเอียดของแต่ละองค์ประกอบดังต่อไปนี้

1. อุปกรณ์คอมพิวเตอร์ คือ เครื่องคอมพิวเตอร์รวมไปถึงอุปกรณ์ต่อพ่วงต่างๆ เช่น Digitizer, Scanner, Plotter, Printer หรืออื่นๆ เพื่อใช้ในการนำเข้าข้อมูล ประมวลผล แสดงผล และผลิตผลลัพธ์ของการทำงาน
2. ซอฟต์แวร์ คือชุดของคำสั่งสำเร็จรูป เช่น โปรแกรม ArcGIS, QGIS, MapInfo เป็นต้น ซึ่งประกอบด้วยพังก์ชันการทำงานและเครื่องมือที่จำเป็นต่างๆ สำหรับนำเข้าและปรับแต่งข้อมูล จัดการฐานข้อมูล เรยิกคัน วิเคราะห์ และจำลองภาพ
3. ข้อมูล คือข้อมูลต่างๆ ที่จะใช้ในระบบ GIS และถูกจัดเก็บในรูปแบบของฐานข้อมูลโดยได้รับการดูแลจากระบบจัดการฐานข้อมูลหรือ DBMS ข้อมูลจะเป็นองค์ประกอบที่สำคัญรองลงมาจากบุคลากร
4. บุคลากร คือ ผู้ปฏิบัติงานซึ่งเกี่ยวข้องกับระบบสารสนเทศภูมิศาสตร์ เช่น ผู้นำเข้าข้อมูล ช่างเทคนิค ผู้ดูแลระบบฐานข้อมูล ผู้เชี่ยวชาญสำหรับวิเคราะห์ข้อมูล ผู้บริหาร ซึ่งต้องใช้ข้อมูลในการตัดสินใจ บุคลากรจะเป็นองค์ประกอบที่สำคัญที่สุดในระบบ GIS เนื่องจากถ้าขาดบุคลากร ข้อมูลที่มีอยู่จำนวนมากมหาศาลนั้นก็จะเป็นเพียงขยะไม่มีคุณค่า ได้เลย เพราะไม่ได้ถูกนำไปใช้งาน อาจจะกล่าวได้ว่าถ้าขาดบุคลากรก็จะไม่มีระบบ GIS
5. วิธีการหรือขั้นตอนการทำงาน คือวิธีการที่องค์กรนั้นๆ นำระบบ GIS ไปใช้งานโดยแต่ละระบบ แต่ละองค์กรยอมรับความแตกต่างกันออกไป ฉะนั้นผู้ปฏิบัติงานต้องเลือกวิธีการในการจัดการกับปัญหาที่เหมาะสมที่สุดสำหรับของหน่วยงานนั้นๆ เอง



รูปที่ 16.1 แสดงองค์ประกอบของ GIS

สำหรับความรู้พื้นฐานเกี่ยวกับระบบ GIS อยู่นอกเหนือหนังสือเล่มนี้ ผู้ที่สนใจสามารถอ่านเพิ่มเติมได้จากหนังสือหรือตำราเกี่ยวกับระบบสารสนเทศภูมิศาสตร์ที่มีจำหน่ายอยู่ทั่วไป หรืออ่านได้จากเว็บไซต์ <http://www.gisthai.org/> การเขียนโปรแกรมกับระบบ ArcGIS ผู้เขียนโปรแกรมต้องมีความรู้เกี่ยวกับการเขียนโปรแกรมเชิงวัตถุด้วย ซึ่งสามารถอ่านได้จากบทที่ 11 ในหนังสือเล่มนี้

ในบทนี้ผู้เขียนมีเป้าหมายเพื่อให้ผู้อ่านสามารถเขียนโปรแกรมฯ พร้อมควบคุมการทำงานของซอฟต์แวร์ในองค์ประกอบข้อที่ 2 ของ GIS ด้านบน ให้สามารถทำงานได้อย่างมีประสิทธิภาพ ดังนั้น ผู้อ่านควรต้องมีความรู้พื้นฐานเกี่ยวกับการใช้งานระบบ GIS (ArcGIS) มาบ้างพอสมควร สำหรับซอฟต์แวร์ที่ผู้เขียนนำมาทดสอบการเขียนโปรแกรมคือ ArcGIS (ESRI) เวอร์ชัน 10.x ซึ่งเป็นซอฟต์แวร์ที่ได้รับความนิยมเป็นอย่างมากในการประมวลผลข้อมูลด้าน GIS ในปัจจุบัน



2. การติดตั้งโปรแกรม ArcGIS

โปรแกรม ArcGIS เป็นซอฟต์แวร์ที่ใช้สำหรับประมวลผลงานด้าน GIS ซึ่งครอบคลุมการทำงานด้าน GIS เกือบทั้งหมด ผู้ใช้งานสามารถดาวน์โหลดและติดตั้งทดลองใช้งานได้ฟรี ประมาณ 60 วัน ได้จาก <http://www.esri.com/software/arcgis/arcgis-for-desktop/free-trial> ปัจจุบันเป็นเวอร์ชันที่ 10.2 ในขั้นตอนก่อนการดาวน์โหลดผู้ใช้ต้องทำการลงทะเบียนเพื่อขอ Authorize number ผ่านทาง email สำหรับขั้นตอนการติดตั้ง ArcGIS เวอร์ชันทดลอง (ArcGIS/Desktop_10xx_xxxxxx.exe) ดังนี้คือ

- ต้องติดตั้ง Microsoft .NET Framework 3.5 SP1 หรือสูงกว่าก่อนเสมอ
- ถ้ามีซอฟต์แวร์ ArcGIS เวอร์ชันเก่า ให้ถอนการติดตั้งเวอร์ชันเดิมออกก่อน
- ดับเบิลคลิกแฟ้มชื่อ ArcGIS/Desktop_10xx_xxxxxx.exe เพื่อทำการขยายแฟ้มที่ใช้ในการติดตั้ง (ให้เก็บแฟ้มที่ขยายใน C:\temp เป็นต้น)
- ดับเบิลคลิกแฟ้ม Setup.exe เพื่อติดตั้ง ArcGIS จะปรากฏหน้าต่างแสดง ลิขสิทธิ์ (license agreement) ให้ผู้ใช้เลือก accept \Rightarrow เลือกติดตั้งแบบสมบูรณ์ (Complete) \Rightarrow คลิก next เพื่อยยอมรับการติดตั้งไปเรื่อยๆ จนกว่าจะเสร็จสิ้นการติดตั้ง

การลงทะเบียนเพื่อขอใช้งาน ArcGIS เวอร์ชันทดลอง

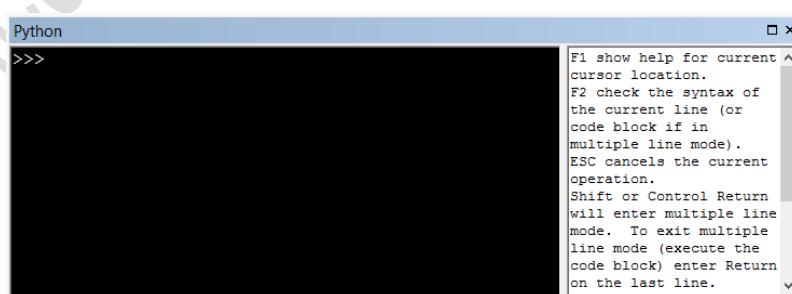
- โปรแกรมจะเปิด ArcGIS administrator ให้ผู้ใช้เลือก ArcGIS for Desktop Advanced (Single Use) และคลิกที่ Authorize Now
- เลือก I have installed my software and need to authorize it และคลิก next
- เลือก Authorize with Esri now using the Internet (คอมพิวเตอร์ต้องเชื่อมต่ออินเทอร์เน็ตด้วย แต่ถ้าไม่ได้เชื่อมต่ออินเทอร์เน็ตให้เลือก authorize ผ่าน email แทนก็ได้) และคลิก next
- ป้อนข้อมูลผู้ใช้งานให้ครบและคลิก next
- ป้อนข้อมูล Authorization number ซึ่งขึ้นต้นด้วย EVXXXXXXXXXX เช่น EVA280525629 (ตัวเลขดังกล่าวได้จากขั้นตอนก่อนการดาวน์โหลดซอฟต์แวร์)
- เมื่อลงทะเบียนแบบ online เสร็จแล้ว (ใช้เวลาประมาณ 30 – 60 วินาที) ให้เลือก finish
- ปิดโปรแกรม ArcGIS administrator, โปรแกรมพร้อมใช้งานแล้ว

สำหรับโปรแกรม ArcGIS รุ่นทดลองสามารถใช้งานซอฟต์แวร์ได้ครบถ้วนฟังชัน แต่ใช้งานได้เพียง 60 วันเท่านั้น

1. การเรียกใช้งาน Python

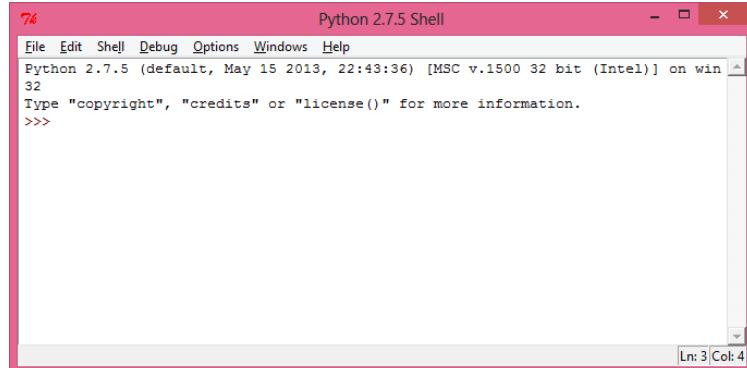
ขณะทำการติดตั้ง ArcGIS เวอร์ชัน 10.x โปรแกรมจะทำการติดตั้ง Python เวอร์ชัน 2.7.x ให้โดยอัตโนมัติ ผู้ใช้งานสามารถเรียกใช้ Python ได้ 2 แบบคือ

- เรียกใช้งานจากภายใน ArcGIS เรียกว่า Python shell window โดยคลิกที่เมนู geoprocessing \Rightarrow Python แต่วิธีนี้จะเป็นแบบ Interactive mode คือ Python จะรับคำสั่งทีละบรรทัดพร้อมกับประมวลผลคำสั่งทันที ดังรูปที่ 16.2



รูปที่ 16.2 Python shell window

- หรือเรียกจากภายในโปรแกรม ArcGIS โดยเลือก Start \Rightarrow Programs \Rightarrow Python 2.7.x \Rightarrow IDLE (Python GUI) สำหรับวินโดวส์ 7 และ 8 กดปุ่ม windows  บนแป้นพิมพ์ \Rightarrow IDLE (Python GUI) ดังรูปที่ 16.3



รูปที่ 16.3 IDLE (Python GUI)

สำหรับวินโดวส์เขียนโปรแกรมสามารถเขียนโปรแกรมในลักษณะเป็นสคริปต์ได้ คือเขียนโปรแกรมหลายๆ คำสั่งรวมเข้าไว้ด้วยกัน เมื่อเขียนโปรแกรมเสร็จแล้วต้องทำการบันทึกเป็นแฟ้มก่อน แล้วจึงสามารถสั่งให้โปรแกรมประมวลผลได้

3. การเขียนโปรแกรมไฟล์อนกับ GIS (ArcGIS 10.x)

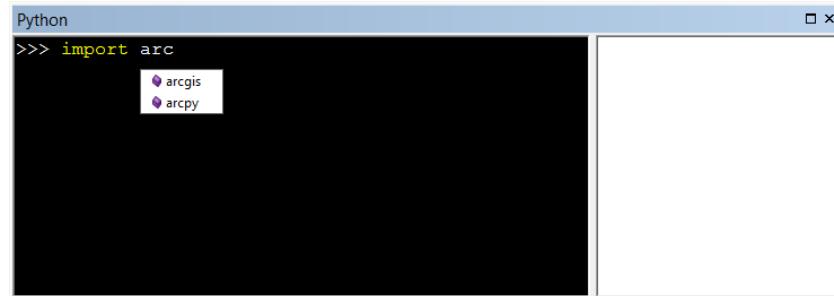
การเขียนสคริปต์สำหรับการประมวลผลข้อมูลทางภูมิศาสตร์ (Geoprocessing script) คือการประมวลผลข้อมูลทางภูมิศาสตร์กับข้อมูลที่มีจำนวนมาก ใช้เวลานาน และต้องทำงานช้าๆ ในรูปแบบเดิมๆ จนกว่าจะได้คำตอบที่ต้องการ การเขียนสคริปต์จะช่วยให้ผู้เขียนโปรแกรมสามารถประยุกต์ทรัพยากรในการประมวลผลลงได้มาก สำหรับ ArcGIS ได้จัดเตรียมชุดคำสั่งหรือโมดูลสำหรับงานด้าน Geoprocessing ไว้ มีชื่อว่า Arcpy site package เพื่อช่วยอำนวยความสะดวกในการจัดการควบคุมและเข้าถึงสภาพแวดล้อมของ ArcGIS

3.1 การเรียกใช้งาน Arcpy site package ด้วยไฟล์

- เปิดแฟ้มเอกสารที่ใช้สำหรับเก็บแผนที่ชื่อ Crime1 (มีส่วนขยายเป็น .mxl) ในไดเรกทอรี C:\PythonData\Ch16 (ให้ทำการคัดลอกไดเรกทอรี PythonData จากแผ่น CD ลงใน C:\PythonData) ด้วยโปรแกรม ArcMap ดังนี้ \Rightarrow เมนู File \Rightarrow Open \Rightarrow C:\PythonData\Ch16\Crime1.mxd
- เปิดโปรแกรม Python window เพื่อเขียนสคริปต์จาก \Rightarrow เมนู Geoprocessing \Rightarrow Python

3. ในหน้าต่างของ Python window ให้ทำการ import arcpy package ดังนี้

```
>>> import arcpy
```

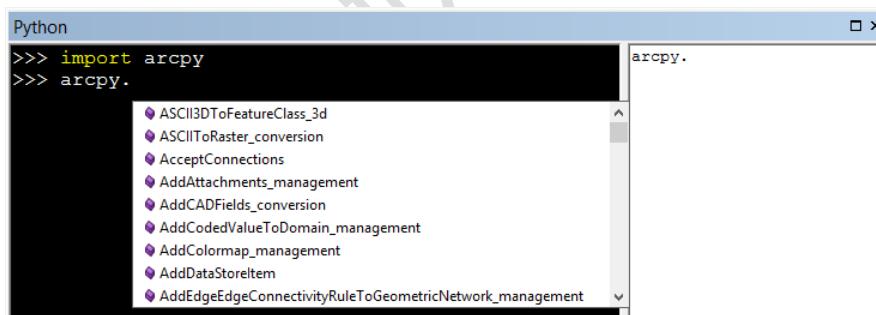


ขณะที่ผู้เขียนโปรแกรมกำลังพิมพ์คำสั่งใดๆ ใน Python window โปรแกรมจะแสดงคำสั่งที่สอดคล้องกับการพิมพ์ของผู้เขียนให้เห็น เพื่ออำนวยความสะดวกในการทำงาน ดังรูปด้านบน

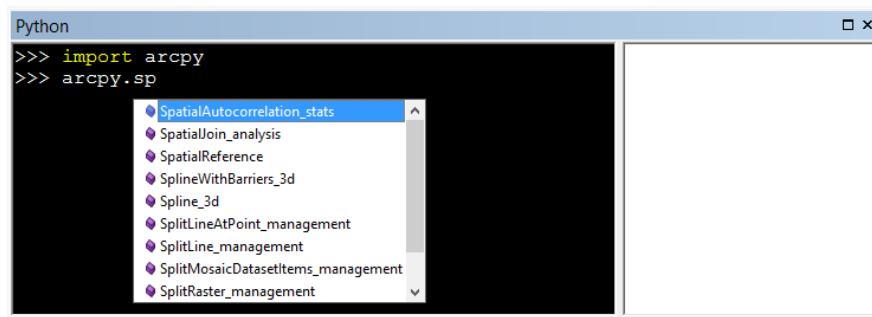


Tips: เมื่อผู้เขียนโปรแกรมพิมพ์คำสั่งใดๆ ใน Python window โปรแกรมจะแสดงคำสั่งที่เกี่ยวข้องอุปกรณ์ให้เห็น ผู้เขียนโปรแกรมสามารถเลือกใช้คำสั่งใดๆ ด้วยการกดปุ่ม Tab หรือเลื่อนเมาส์ไปที่คำสั่งที่ต้องการแล้วคลิกเลือกคำสั่งนั้นๆ ก็ได้

4. เมื่อทำการ import arcpy package เข้ามาในโปรแกรมแล้ว ผู้เขียนโปรแกรมสามารถเรียกใช้งานแอตทริบิวต์, เมธอด, geoprocessing tools, พังชัน หรือคลาสในโปรแกรม ArcGIS ได้ทั้งหมด โดยใช้สัญลักษณ์ . เพราะ arcpy package ถูกเขียนขึ้นจากแนวคิดการโปรแกรมเชิงวัตถุนั่นเอง



จากรูปตัวอย่างด้านบน เมื่อพิมพ์ arcpy. โปรแกรมจะแสดงแอตทริบิวต์, เมธอด, พังชัน, geoprocessing tools และคลาสทั้งหมดใน ArcGIS (ขึ้นอยู่กับ license ที่ติดตั้งว่าเป็นประเภท Basic, Standard หรือ Advanced) ให้ผู้เขียนโปรแกรมเลือกใช้งาน โดยการกดปุ่ม Tab หรือเลื่อนเมาส์ไปยังคำสั่งที่ต้องการก็ได้ ดังรูปด้านล่าง



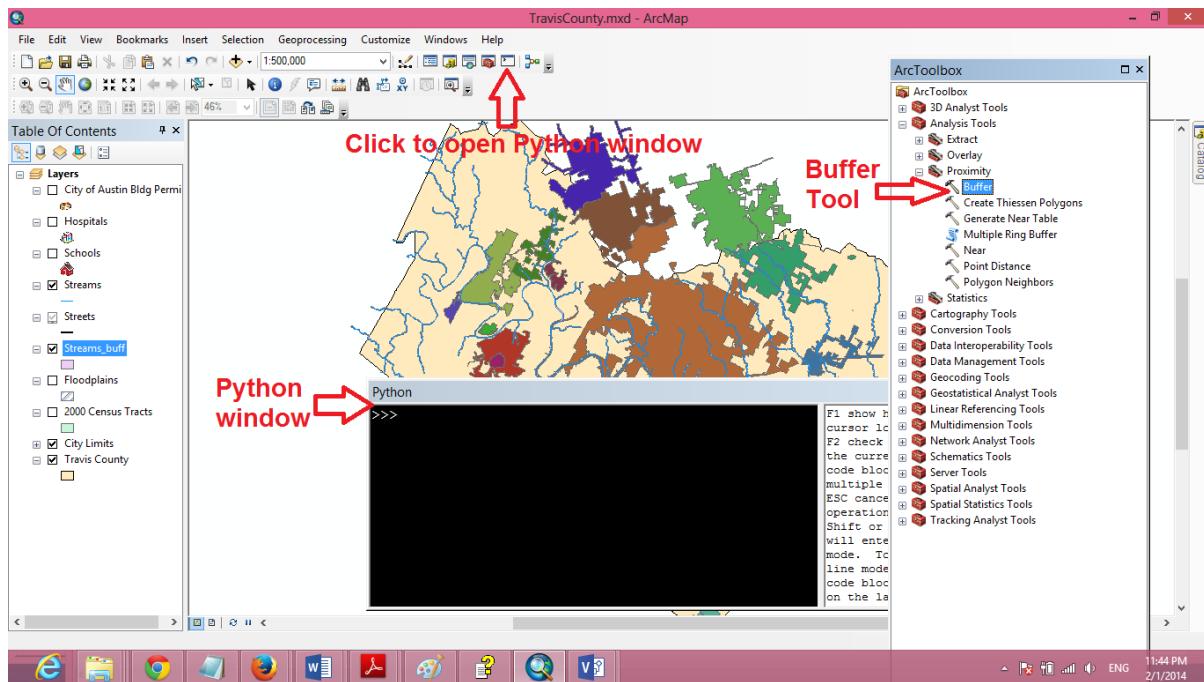
3.2 การเรียกใช้ Tools ใน ArcGIS ด้วยไฟรอน

การประมวลผลข้อมูลทางภูมิศาสตร์ (geoprocessing tasks) อาศัยเครื่องมือที่อยู่ใน ArcToolbox ซึ่งอยู่ในเมนู Geoprocessing เช่น Analysis, Network Analyst และ Spatial Analyst Tool เป็นต้น ในหัวข้อนี้จะใช้ไฟรอนเรียกใช้งาน Tools ที่ชื่อว่า Buffer ใน ArcToolbox

1. เปิดแฟ้มเอกสารที่ใช้สำหรับเก็บแผนที่ชื่อ TravisCounty.mxd ในไดเรคทรอรี C:\PythonData\Ch16 ด้วยโปรแกรม ArcMap
2. เปิดโปรแกรม Python window และทำการ import arcpy package ด้วยคำสั่ง

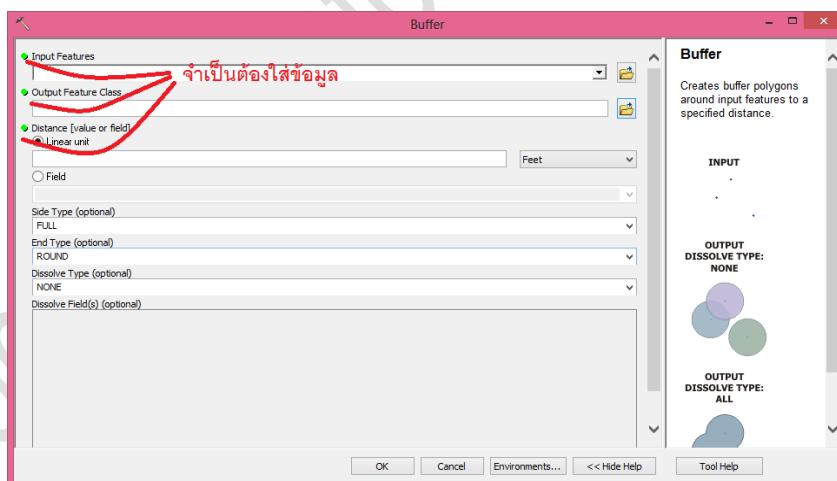

```
>>> import arcpy
```
3. ทำการกำหนดไดเรคทรอรีใช้งาน (current working directory) ด้วยไฟรอนเพื่อใช้เก็บข้อมูลอินพุตและเอาต์พุตที่เกิดจาก ArcMap (เมื่อใช้งานผ่าน ArcMap เลือกเมนู Geoprocessing \Rightarrow Environments \Rightarrow Workspace \Rightarrow Current workspace) ด้วยคำสั่งดังนี้


```
>>> arcpy.env.workspace = "c:\PythonData\workspace"
```
4. ในตัวอย่างนี้จะเลือกเครื่องมือชื่อ buffer สำหรับใช้กับแม่น้ำ (Streams) ในแผนที่ชื่อ TravisCounty โดยเครื่องมือดังกล่าวอยู่ในเมนู Geoprocessing \Rightarrow ArcToolbox \Rightarrow Analysis Tools \Rightarrow Proximity \Rightarrow Buffer ดังรูปที่ 16.4



รูปที่ 16.4 แสดง Python window และ Buffer Tool ใน ArcMap

- ขั้นตอนต่อไปทำการดับเบิลคลิกที่ Buffer Tool ดังแสดงในรูปที่ 16.5 จะเห็นได้ว่าใน Buffer Tool จะมีพารามิเตอร์ที่จำเป็นต้องกำหนดหลายตัว (เป็นจุดสีเขียว) เช่น Input Features, Output Feature Class และ Distance



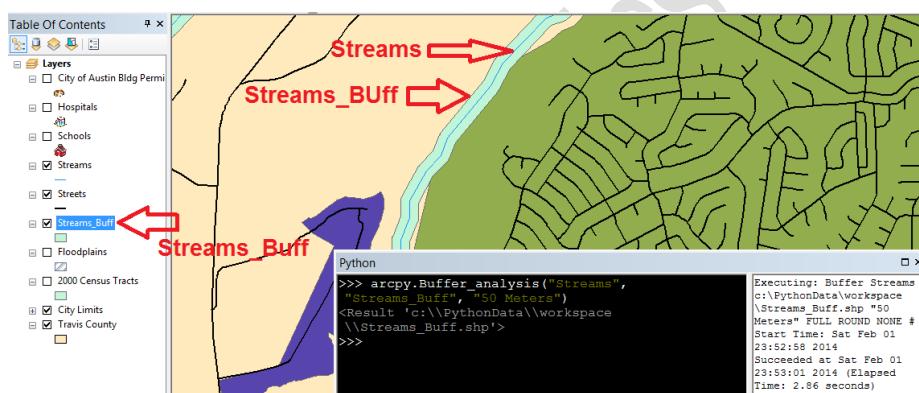
รูปที่ 16.5 แสดงพารามิเตอร์ที่จำเป็นต้องกำหนดใน Buffer Tool

- ปิดโปรแกรม Buffer Tool เพื่อเปลี่ยนมาใช้ไฟชอนสคริปต์สำหรับกำหนดพารามิเตอร์ใน Tool ดังกล่าวแทน โดยใช้ Buffer ข้อนทับข้อมูลของแม่น้ำ (Streams) ในรัศมี 50 เมตร เอาจริงๆ ว่า Streams_Buff ในลักษณะของ polygon ด้วยคำสั่งดังนี้
- ```
>>> arcpy.Buffer_analysis("Streams", "Streams_Buff", "50 Meters")
```

จากคำสั่งข้างบน โปรแกรมเริ่มจากการอ้างถึง arcpy package ด้วยคำสั่ง arcpy ลำดับต่อไป โปรแกรมเรียกใช้ Tool ชื่อว่า Buffer ออยู่ภายใต้เครื่องมือชื่อ Analysis Tools (ใช้ชื่อแทนว่า analysis สามารถดูข้อมูลได้โดยการคลิกขวาที่ Analysis Tools  $\Rightarrow$  เลือก Properites  $\Rightarrow$  ในช่อง Alias) โดยอ้าง ด้วยคำสั่ง Buffer\_analysis (มีรูปแบบการอ้างอิงคือ <toolname>\_<toolbox\_alias>) ภายในวงเล็บ พารามิเตอร์ตัวแรกคือ Input Features มีค่าเท่ากับ "Streams", พารามิเตอร์ตัวที่สองคือ Output Feature Class มีค่าเท่ากับ "Streams\_Buff" และพารามิเตอร์ตัวที่สามคือ Distance มีค่าเท่ากับ "50 Meters" ตามลำดับ

```
Arpy package Tool name Toolbox(alias) Input Features Output Feature Class Distance
arpy.Buffer_analysis("Streams", "Streams_Buff", "50 Meters")
(Buffer Tool)
```

7. ลำดับถัดไปให้ผู้เขียนโปรแกรมใช้ zoom in และ zoom out ในโปรแกรม ArcMap ตรวจสอบว่า Streams\_Buff ถูกสร้างขึ้นจริงหรือไม่ ดังรูปที่ 16.6



รูปที่ 16.6 แสดงการสร้าง Streams\_Buff

8. เห็นได้ว่าการเขียนสคริปต์ควบคุม ArcMap มีหลายขั้นตอนและในบางงานอาจจะมี ความซับซ้อน ผู้เขียนแนะนำให้ผู้ใช้งานออกแบบการทำงานในลักษณะเป็น Flowchart ที่แสดงการทำงานอย่างคร่าวๆ ก่อน หลังจากนั้นจึงเขียนโปรแกรมไฟร์อนตาม Flowchart ที่ออกแบบไว้



**Note:** ผู้เขียนโปรแกรมสามารถดูรายละเอียดเกี่ยวกับการใช้งานคำสั่งต่างๆ โดยใช้ ArcGIS Desktop Help เช่น ต้องการแสดงรายละเอียดเกี่ยวกับคำสั่ง Buffer ได้จากเมนู Help ใน ArcMap  $\Rightarrow$  ArcGIS Desktop Help  $\Rightarrow$  Contents tab  $\Rightarrow$  Desktop  $\Rightarrow$  Geoprocessing  $\Rightarrow$  Tool Reference  $\Rightarrow$  Alalysis Toolbox  $\Rightarrow$  Proximity Toolset  $\Rightarrow$  Buffer

### 3.3 การสร้างตัวแปรเพื่อใช้งานร่วมกับ arcpy

การเขียนโปรแกรมร่วมกับ Toolboxes ใน ArcGIS มีความจำเป็นต้องสร้างตัวแปรเพื่อรับผลลัพธ์ที่เกิดจาก Tools ต่างๆ หลังจากทำงานเสร็จแล้ว หรือใช้สำหรับเป็นพารามิเตอร์ให้กับเมธอดที่เรียกใช้งาน ในหัวข้อนี้จะอธิบายเกี่ยวกับการสร้างตัวแปรเพื่อใช้งานตามที่กล่าวมาแล้ว

1. เป็นโปรแกรม IDLE สำหรับเขียนโปรแกรมไฟชอนจากการงานนอก ArcMap
2. ตั้งชื่อสคริปต์ว่า DealWithVariables.py และบันทึกลงใน C:\PythonData\Ch16
3. ภาษาในสคริปต์ให้พิมพ์คำสั่งดังนี้

```
import arcpy
```
4. สร้างตัวแปรชื่อ path สำหรับเก็บข้อมูลของอินพุตและเอาต์พุต ซึ่งอยู่ในไดเรคทรอร์ (ถ้าไม่มีไดเรคทรอร์ตั้งกล่าวให้ผู้เขียนโปรแกรมสร้างไดเรคทรอร์ขึ้นก่อน)

```
path = "c:\PythonData\data"
```
5. กำหนดค่าในตัวแปร path ให้กับ env.workspace

```
arcpy.env.workspace = path
```
6. เรียกใช้งานเมธอด ListFields เพื่อแสดง fields ทั้งหมดในแฟ้มชื่อ "Building\_Permits.shp" (.shp หรือ Shapefile คือไฟล์ที่เก็บข้อมูลเวคเตอร์และชั้นข้อมูล) ค่าที่ส่งกลับจากเมธอด ListFields จะเก็บไว้ในตัวแปรชื่อ Fields

```
fields = arcpy.ListFields("Building_Permits.shp")
```
7. ใช้ for เพื่อนำข้อมูลที่อยู่ในตัวแปร fields แต่ละตัวเก็บไว้ในตัวแปรชื่อ fld

```
for fld in fields:
```
8. พิมพ์ค่าที่อยู่ใน fld แต่ละค่าออกมาระดับผลด้วยคำสั่ง print

```
print fld.name หรือ print(fld.name)
```
9. คำสั่งทั้งหมดที่กล่าวมาถึงข้างต้นนี้แสดงได้ดังนี้

```
import arcpy
path = "c:\PythonData\data"
arcpy.env.workspace = path
fields = arcpy.ListFields("Building_Permits.shp")
for fld in fields:
 print fld.name
```
10. บันทึกโปรแกรมพร้อมกับสั่งให้โปรแกรมประมวลผลโดยการกด F5



**OUTPUT**

|            |
|------------|
| FID        |
| Shape      |
| LOC_NAME   |
| STATUS     |
| SCORE      |
| SIDE       |
| ...        |
| PROPERTYRS |
| PARENTRSN  |
| Work_Desc2 |

### 3.4 การเรียกใช้งานโมดูลใน arcpy package

ใน arcpy package มีโมดูลที่สำคัญหลายโมดูล เช่น โมดูลจัดการเกี่ยวกับแผนที่ (Maping Module) ชื่อว่า arcpy.mapping, โมดูลเกี่ยวกับการเข้าถึงข้อมูล (Data Access Module) ชื่อว่า arcpy.da, โมดูลวิเคราะห์ข้อมูลเชิงพื้นที่ (Spatial Analyst Module) ชื่อว่า arcpy.sa, โมดูลการวิเคราะห์ทางธรณีสถิติ (Geostatistical module) ชื่อว่า arcpy.ga, โมดูลการวิเคราะห์แผนภาพเครือข่าย (Network Analyst module) และโมดูลเกี่ยวกับเวลา (Time module) ชื่อว่า arcpy.time เป็นต้น ในหัวข้อนี้จะสาธิตการเรียกใช้โมดูลชื่อ arcpy.mapping

1. เปิดแฟ้มข้อมูลชื่อว่า Crime1.mxd ในไดเรคทรอร์ C:\PythonData\Ch16 ด้วยโปรแกรม ArcMap

2. เปิดโปรแกรม Python window และทำการ import arcpy.mapping โดยสร้างชื่อเรียกแทน (alias) ใหม่ว่า mapping เพื่อให้การเรียกใช้งานสั้นและง่ายขึ้น  
import arcpy.mapping as mapping

3. กำหนดให้พythonเรียกใช้แผนที่ที่เปิดใช้งานอยู่ในปัจจุบันชื่อว่า Crime1.mxd โดยใช้สตริง "CURRENT" ผลลัพธ์ที่ส่งกลับคือตัวแหน่งอ้างอิงแผนที่ Crime1 และเก็บไว้ในตัวแปรชื่อ mxd

```
mxd = mapping.MapDocument("CURRENT")
```

4. เรียกใช้เมธอด ListLayers ในโมดูล mapping เพื่อแสดงรายการ Layers ทั้งหมดในแฟ้มของแผนที่ชื่อ Crime1.mxd

```
print mapping.ListLayers(mxd)
```

5. เมื่อพิมพ์คำสั่งในขั้นตอนที่ 4 และกดปุ่ม enter โปรแกรมจะแสดงรายชื่อของ Layers ในแฟ้มแผนที่ Crime1 ดังนี้



OUTPUT

```
[<map layer u'All Crimes in 2009'>, <map layer u'Burglaries in 2009'>, <map layer u'Crime Density by School District'>, <map layer u'Bexar County Boundary'>, <map layer u'Test Performance by School District'>, <map layer u'Bexar County Boundary'>, <map layer u'Bexar County Boundary'>, <map layer u'Texas Counties'>, <map layer u'School Districts'>, <map layer u'Crime Surface'>, <map layer u'Bexar County Boundary'>]
```

จากເອົາຕຸພູດຕ້ານບັນແນນທີ່ໃຊ້ Crime1 ມີຮາຍການຂອງເລເຢອຣ໌ທັງໝົດ 11 ເລເຢອຣ໌ ເຊັ່ນ All

Crimes in 2009, Buglaries in 2009 และ Crime Density by School District เป็นต้น

### 3.5 การอ้างอิงแผนที่ที่ใช้งานภายใน ArcMap

เมื่อผู้เขียนโปรแกรมต้องการดำเนินการใดๆ กับแผนที่ จำเป็นต้องอ้างอิงผ่านชื่อของแผนที่นั้นๆ ก่อนเสมอ เมื่อเขียนโปรแกรมด้วย Python ใน Python window จะต้องใช้คีย์เวิร์ด "CURRENT" เพื่ออ้างอิงแผนที่ปัจจุบัน แต่ถ้าเขียนโปรแกรม Python จากภายนอกโปรแกรม ArcMap จะไม่สามารถใช้คีย์เวิร์ด "CURRENT" ใน การอ้างถึงแผนที่ได้ ในหัวข้อนี้จะสาธิตการเขียนโปรแกรม Python เพื่อแก้ไขคุณสมบัติต่างๆ ในแผนที่กับคีย์เวิร์ด "CURRENT"

1. เปิดแฟ้มชื่อ Crime2.mxd ในไดเรคทรอรี C:\PythonData\Ch16 ด้วยโปรแกรม ArcMap
2. เปิดโปรแกรม Python window และทำการนำเข้าโมดูล mapping เข้ามายังงาน

```
import arcpy.mapping as mapping
```
3. ทำการอ้างอิงไปยังแผนที่ชื่อ Crime2 ผ่านคลาส MapDocument และเก็บผลลัพธ์จาก การอ้างอิงไว้ในตัวแปรชื่อ mxd

```
mxd = mapping.MapDocument("CURRENT")
```
4. แสดง title ของแผนที่ Crime2 โดยใช้คำสั่ง print

```
print mxd.title
```
5. ทดสอบกำหนด title ใหม่ให้กับแผนที่ Crime2 เป็น "New Crime Project"

```
mxd.title = "New Crime Project"
```
6. ทดสอบโดยการบันทึกแฟ้มแผนที่ใหม่เป็นชื่อ NewCrime2 ด้วยเมธอด saveACopy

```
mxd.saveACopy("c:\PythonData\Ch16\NewCrime2.mxd")
```
7. ทดสอบรันสคริปต์ดังกล่าว และสังเกตุผลลัพธ์ที่ได้ โดยเปิดแฟ้ม NewCrime2 แล้วเลือกเมนู File  $\Rightarrow$  Map Document Properties  $\Rightarrow$  สังเกตุในส่วนของ Title จะเปลี่ยน จาก "Crime Project" เป็น "New Crime Project"

### 3.6 การอ้างอิงแผนที่ที่ใช้งานภายนอก ArcMap

1. เปิดไฟรอน IDLE (ซึ่งอยู่ภายนอกโปรแกรม ArcMap) จาก Start  $\Rightarrow$  Programs  $\Rightarrow$  ArcGIS  $\Rightarrow$  Python 2.7  $\Rightarrow$  IDLE
2. สร้างสคริปต์ใหม่โดยเลือกเมนู File  $\Rightarrow$  New Window
3. นำเข้า arcpy.mapping และเปลี่ยนชื่อเป็น mapping

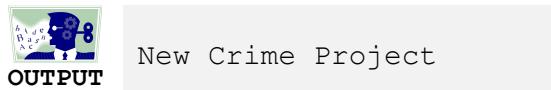
```
import arcpy.mapping as mapping
```
4. ทำการอ้างอิงไปยังแผนที่ชื่อ NewCrime2.mxd ในไดเรคทรอรี C:\PythonData\Ch16 ผ่านคลาส MapDocument

```
mxd =
```

```
mapping.MapDocument("C:\PythonData\Ch16\NewCrime2.mxd")
```
5. ทดสอบโดยการพิมพ์-export ทริบิวต์ Title ของ NewCrime2 ด้วยคำสั่ง print

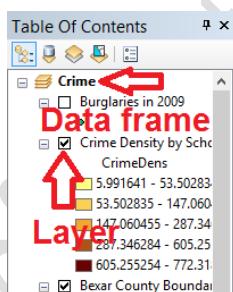
```
print mxd.title
```

- บันทึกแฟ้มชื่อ ReferMapOutsideArcGIS.py และรันโปรแกรม ผลลัพธ์ที่ได้คือ



### 3.7 การแสดงเฟรมข้อมูล (Data frame)

ในส่วนของ Table of Contents (ด้านซ้ายมือ) ของ ArcMap จะแสดงรายละเอียดของเฟรมข้อมูล (data frame) และภายใต้แต่ละเฟรมจะประกอบไปด้วยเลเยอร์ (Layers) และตาราง (Tables) ดังรูป ผู้เขียนโปรแกรมสามารถดึงรายการของเฟรมข้อมูลมาแสดงผลได้ โดยใช้เมธอด ListDataFrame และใช้เมธอด ListLayers สำหรับดึงเลเยอร์ทั้งหมดที่อยู่ในเฟรมข้อมูลที่ต้องการมาแสดงผล สำหรับการแสดงผลตารางจะใช้เมธอดชื่อว่า ListTables ได้เช่นเดียวกัน ในหัวข้อนี้จะสาธิตการดึงรายการเฟรมข้อมูลมาแสดงผล



- เปิดแฟ้มชื่อ Crime2.mxd ในไดเรกทรอรี C:\PythonData\Ch16 ด้วยโปรแกรม ArcMap
- เปิดโปรแกรม Python window และทำการนำเข้าโมดูล mapping เข้ามาทำงาน

```
import arcpy.mapping as mapping
```
- ทำการอ้างอิงไปยังแผนที่ชื่อ Crime2 ผ่านคลาส MapDocument และเก็บผลลัพธ์จาก
การอ้างอิงไว้ในตัวแปรชื่อ mxd

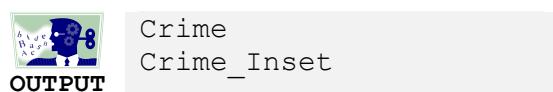
```
mxd = mapping.MapDocument("CURRENT")
```
- แสดงรายการเฟรมข้อมูลด้วยเมธอด ListDataFrames โดยเลือกเฉพาะชื่อเฟรมที่ขึ้นต้น
ด้วยตัวอักษร C ตัวใหญ่เท่านั้น รายชื่อเฟรมที่ได้จะเก็บไว้ในตัวแปรชื่อ frames
เมธอด ListDataFrames ต้องการพารามิเตอร์ 2 ตัว คือ ตัวแปรที่อ้างอิงแผนที่ (mxd)
และชื่อเฟรมข้อมูลที่ต้องการแสดงผล ถ้าไม่กำหนดชื่อเฟรมข้อมูล โปรแกรมจะแสดงชื่อ
เฟรมทั้งหมดที่อยู่ใน Table of Contents

```
frames = mapping.ListDataFrames(mxd, "C*")
```

5. ค่าที่ส่งกลับมาจากการเมธอด ListDataFrames คือรายการเฟรมข้อมูลจาก Table of Contents ใน ArcMap ดังนั้นจำเป็นต้องใช้ for เพื่อวนลูปดึงข้อมูลแต่ละรายการใน frames มาเก็บไว้ในตัวแปร df และแสดงผลทีละค่าในแต่ละรอบของ for loop ดังนี้
- ```
for df in frames:
    print df.name
```

df.name คือการอ้างอิงถึงแอ็ตทริบิวต์ name ของอ็อปเจกต์ Data Frame

6. เมื่อสั่งรันโปรแกรมผลลัพธ์ที่ได้คือ



3.8 การแสดงเลเยอร์ (Layers)

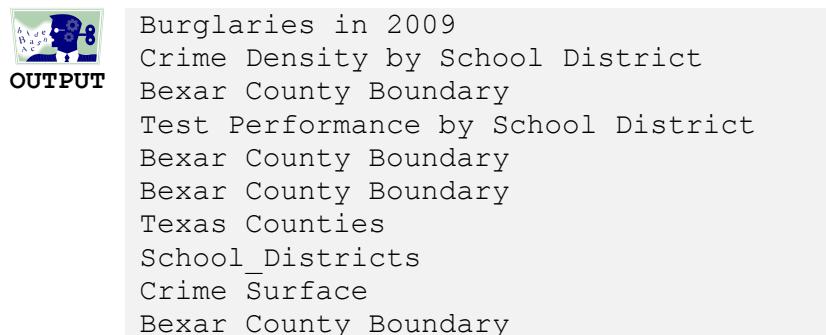
1. เปิดแฟ้มชื่อ Crime2.mxd ในไดเรคทรอรี่ C:\PythonData\Ch16 ด้วยโปรแกรม ArcMap อีกครั้ง
2. เปิดโปรแกรม Python window และทำการนำเข้าโมดูล mapping เข้ามารаХาน

```
import arcpy.mapping as mapping
```
3. ทำการอ้างอิงไปยังแผนที่ชื่อ Crime2 ผ่านคลาส MapDocument และเก็บผลลัพธ์จากการอ้างอิงไว้ในตัวแปรชื่อ mxd

```
mxd = mapping.MapDocument("CURRENT")
```
4. แสดงรายการเลเยอร์ด้วยเมธอด ListLayers และเก็บไว้ในตัวแปรชื่อ layers

```
layers = mapping.ListLayers(mxd)
พารามิเตอร์ mxd คือตัวแปรที่อ้างอิงไปยังแผนที่ Crime2
```
5. ค่าที่ส่งกลับมาจากการเมธอด ListLayers คือรายการเลเยอร์ทั้งหมดที่อยู่ใน Table of Contents ดังนั้นจำเป็นต้องใช้ for เพื่อวนลูปดึงข้อมูลแต่ละรายการในตัวแปร layers มาเก็บไว้ในตัวแปร lyr และแสดงผลทีละค่าในแต่ละรอบของ for loop ดังนี้

```
for lyr in layers:
    print lyr.name
```
6. เมื่อสั่งรันโปรแกรมผลลัพธ์ที่ได้คือ



3.9 การคัดกรองเฉพาะเลเยอร์ที่ต้องการ

ในหัวข้อที่ผ่านมาได้สาธิตวิธีการดึงข้อมูลของเลเยอร์ทั้งหมดจาก Table of Contents มาแสดงผล สำหรับในหัวข้อนี้จะสาธิตการคัดกรองเลเยอร์ที่ผู้เขียนโปรแกรมต้องการใช้งานเท่านั้นมาแสดง

1. เปิดแฟ้มชื่อ Crime2.mxd และทำการนำเข้าโมดูล mapping พร้อมกับกำหนดให้ใช้แทนที่ปัจจุบันที่โปรแกรม ArcMap กำลังเปิดใช้งานอยู่


```
import arcpy.mapping as mapping
mxd = mapping.MapDocument("CURRENT")
```
2. แสดงรายการเฟรมข้อมูลด้วยเมธอด ListDataFrames โดยเลือกเฉพาะเฟรมชื่อ Crime เข้ามาทำงานเท่านั้น (ซึ่งภายในเฟรมดังกล่าวมีเลเยอร์ทั้งหมด 3 เลเยอร์คือ Burglaries in 2009, Crime Density by School District และ Bexar County Boundary) โดยใช้ if ในการตรวจสอบเงื่อนไขเพื่อกรองเฉพาะเฟรมที่ชื่อ Crime


```
for df in mapping.ListDataFrames(mxd):
    if (df.name == 'Crime'):
```
3. เมื่อเลือกเฉพาะเฟรมข้อมูลที่ชื่อ Crime และ โปรแกรมจะส่งที่อยู่สำหรับอ้างอิงเฟรมเก็บไว้ในตัวแปร df ขั้นตอนต่อไปจะนำเอาออบเจกต์ df เป็นพารามิเตอร์ให้กับเมธอด ListLayers อีกครั้ง เพื่อเลือกเฉพาะเลเยอร์ที่ผู้เขียนโปรแกรมต้องการ (ในที่นี่จะเลือกเลเยอร์ที่ขึ้นต้นด้วย 'Burg*') โดยระบุชื่อเลเยอร์ที่ต้องการเป็นพารามิเตอร์ให้กับเมธอด ListLayers ดังนี้


```
layers = mapping.ListLayers(mxd, 'Burg*', df)
```

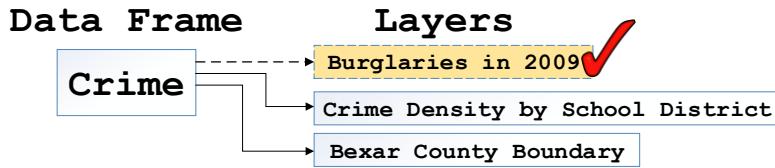


Tips: * แทนตัวอักษรหรือตัวเลขใดๆ ก็ได้ เช่น Burg* หมายถึง Burglaries, Burger, Burg16, Burg_ หรือ Burg_lar ก็ถูกต้องทั้งหมด

4. ค่าที่เก็บอยู่ในตัวแปร layers คือรายการเลเยอร์ทั้งหมดที่ขึ้นต้นด้วยคำว่า 'Burg' ดังนั้นจำเป็นต้องใช้ for เพื่อวนลูปดึงข้อมูลแต่ละรายการในตัวแปร layers มาเก็บไว้ในตัวแปร lyr และแสดงผลที่ละค่าในแต่ละรอบของ for loop ดังนี้


```
for lyr in layers:
    print lyr.name
```
5. เมื่อนำมาสั่งทั้งหมดตั้งแต่ขั้นตอนที่ 1 ถึง 4 มาเขียนเป็นสคริปต์จะได้ดังนี้


```
import arcpy.mapping as mapping
mxd = mapping.MapDocument("CURRENT")
for df in mapping.ListDataFrames(mxd):
    if (df.name == 'Crime'):
        layers = mapping.ListLayers(mxd, 'Burg*', df)
        for layer in layers:
            print layer.name
```
6. เมื่อสั่งรันโปรแกรมผลลัพธ์ที่ได้ดังนี้



3.10 การปรับเปลี่ยนขอบเขตของแผนที่

ในบางสถานะการณ์อาจจำเป็นต้องมีการปรับเปลี่ยนขอบเขตของแผนที่ เช่น ในกรณีที่ผู้ใช้งานทำการสร้างแผนที่มากกว่า 1 แผนที่ และแต่ละแผนที่มีคุณสมบัติที่แตกต่างกัน สำหรับการปรับเปลี่ยนขอบเขตสามารถทำได้หลายวิธี แต่ในหัวข้อนี้จะใช้แอ็ตทริบิวต์ DefinitionQuery, extent และเมธอด GetExtent ที่อยู่ในคลาส DataFrame

1. เปิดแฟ้มชื่อ Crime2.mxd และทำการนำเข้าโมดูล mapping พร้อมกับกำหนดให้ใช้แผนที่ปัจจุบันที่โปรแกรม ArcMap กำลังเปิดใช้งานอยู่

```
import arcpy.mapping as mapping
mx = mapping.MapDocument("CURRENT")
```

2. แสดงรายการเฟรมข้อมูลด้วยเมธอด ListDataFrames โดยเลือกเฉพาะเฟรมชื่อ Crime เข้ามาทำงานเท่านั้นด้วยการใช้คำสั่ง if ตรวจสอบเงื่อนไข

```
for df in mapping.ListDataFrames(mx):
    if (df.name == 'Crime'):
```

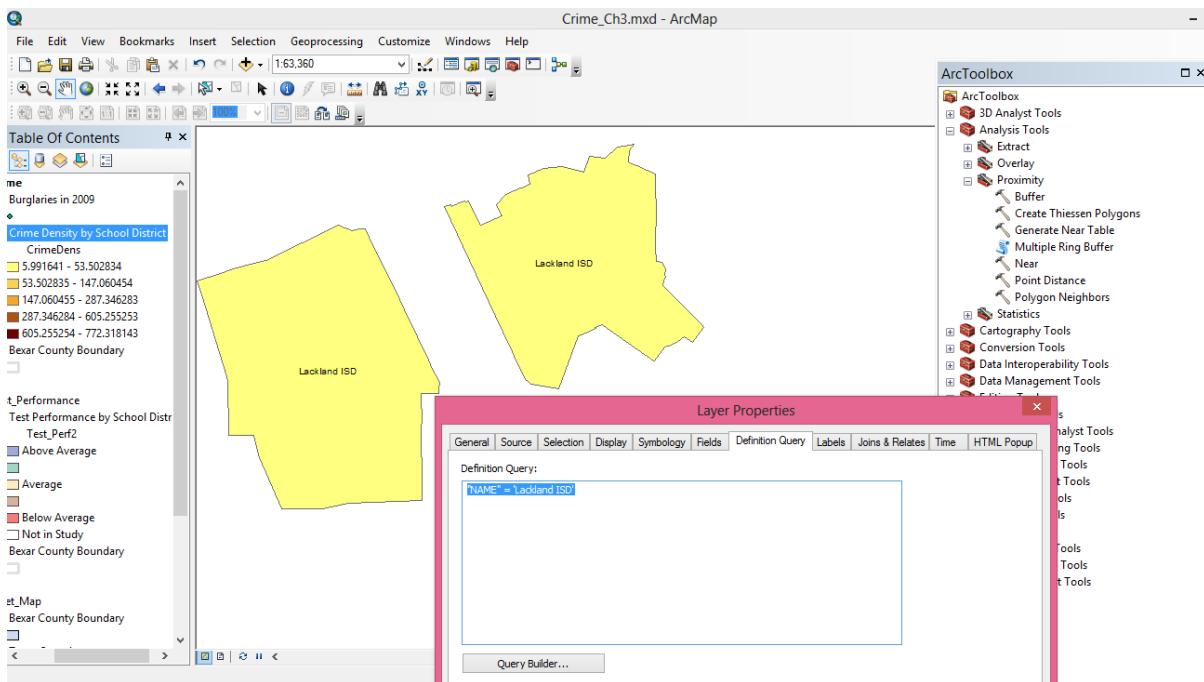
3. เมื่อเลือกเฉพาะเฟรมข้อมูลที่ชื่อ Crime และ จำกัดนั้นให้เมธอด ListLayers เพื่อเลือกเฉพาะเลเยอร์ชื่อว่า 'Crime Density by School District' เท่านั้น ดังนี้

```
layers = mapping.ListLayers(mx, 'Crime Density by School District', df)
```

4. ค่าที่เก็บอยู่ในตัวแปร layers มีเพียงเลเยอร์เดียวคือ Crime Density by School District แต่จำเป็นต้องใช้ for เพื่อดึงข้อมูลในตัวแปร layers มาทำงาน ดังนี้

```
for layer in layers:
    query = '"NAME" = \'Lackland ISD\''
    layer.definitionQuery = query
    df.extent = layer.getExtent()
```

สำหรับตัวแปร query จะเก็บสตริงเท่ากับ "NAME" = 'Lackland ISD' และกำหนดให้กับแอ็ตทริบิวต์ definitionQuery ของคลาสเลเยอร์ จำกัดนั้นโปรแกรมจะสั่งให้เลเยอร์สร้างแผนที่ใหม่โดยแสดงเฉพาะ Lackland ISD เท่านั้น ด้วยคำสั่ง getExtent() เมื่อสั่งให้รันโปรแกรม ผลการทำงานแสดงดังในรูปที่ 16.7



รูปที่ 16.7 แสดงการปรับเปลี่ยนข้อมูลของแผนที่

5. เขียนเป็นสคริปต์ทั้งหมดได้ดังนี้

```
import arcpy.mapping as mapping
mx = mapping.MapDocument("CURRENT")
for df in mapping.ListDataFrames(mx):
    if (df.name == 'Crime'):
        layers = mapping.ListLayers(mx, 'Crime Density by
School District', df)
        for layer in layers:
            query = '"NAME" = \'Lackland ISD\''
            layer.definitionQuery = query
            df.extent = layer.getExtent()
```

3.11 แสดงรายชื่อตาราง (Tables)

ในโมดูล arcpy.mapping มีเมธอดชื่อว่า ListTableViews สำหรับแสดงรายการของตารางในแผนที่

1. เปิดแฟ้มชื่อ Crime2.mxd และทำการโหลดโมดูล mapping พร้อมกับกำหนดให้ใช้แผนที่ปัจจุบันที่โปรแกรม ArcMap กำลังเปิดใช้งานอยู่

```
import arcpy.mapping as mapping
mx = mapping.MapDocument("CURRENT")
```

2. แสดงรายชื่อตารางด้วยเมธอด ListTableViews ในเอกสารแผนที่ (map document)

```
for tableView in mapping.ListTableViews(mx):
    print tableView.name
```

สั้นรันโปรแกรม ผลลัพธ์ที่ได้ดังนี้



Note: เมื่อต้อง ListTableViews ทำงานได้กับ map document และ data frame เท่านั้น ไม่สามารถใช้งานกับ Layer ได้

3.12 การเพิ่มเลเยอร์ในเอกสารแผนที่ โดยใช้ AUTO_ARRANGE

โดยดูคลาส arcpy.mapping ได้เตรียมเมธอดที่ใช้สำหรับเพิ่มเลเยอร์เข้าไปในเอกสารแผนที่โดยใช้เมธอดชื่อว่า AddLayer() ซึ่งมีรูปแบบการใช้งานดังนี้

1. เปิดแฟ้มชื่อ Crime2.mxd และทำการโหลดโมดูล mapping พร้อมกับกำหนดให้ใช้แผนที่ปัจจุบันที่โปรแกรม ArcMap กำลังเปิดใช้งานอยู่

```
import arcpy.mapping as mapping
mxd = mapping.MapDocument("CURRENT")
```

2. เริ่มต้นโดยการเรียกใช้เมธอด ListDataFrames() เพื่อขอดูเฟรมข้อมูลทั้งหมดที่อยู่ในแผนที่ ผลจากการเรียกเมธอดดังกล่าวจะทำให้ได้รายการของเฟรมข้อมูลจากเอกสารแผนที่ Crime2 คือ Crime, Test_Performance, Inset_Map และ Crime_Inset ในตัวอย่างนี้ต้องการเพิ่มเลเยอร์ใหม่ชื่อว่า School_District ลงในเฟรมข้อมูลชื่อ Crime แบบอัตโนมัติ ("AUTO_ARRANGE") การอ้างอิงข้อมูลจะใช้เมธอด

```
mapping.ListDataFrames(mxd)[0] หมายถึง เฟรมข้อมูลตัวแรกที่คืนมาจากการเรียกใช้เมธอด ListDataFrames ในที่นี้หมายถึงเฟรมข้อมูล Crime ถ้าเรียกใช้เมธอด mapping.ListDataFrames(mxd)[1] หมายถึงเฟรมข้อมูล Test_Performance ตามลำดับ
```

```
df = mapping.ListDataFrames(mxd) [0]
```

3. อ้างอิงไปยังแฟ้มที่เก็บข้อมูลของเลเยอร์ที่ต้องการเพิ่มชื่อ School_District.lyr เก็บอยู่ในไดเรกทรอรี C:\PythonData\Data ด้วยเมธอด Layer ดังนี้

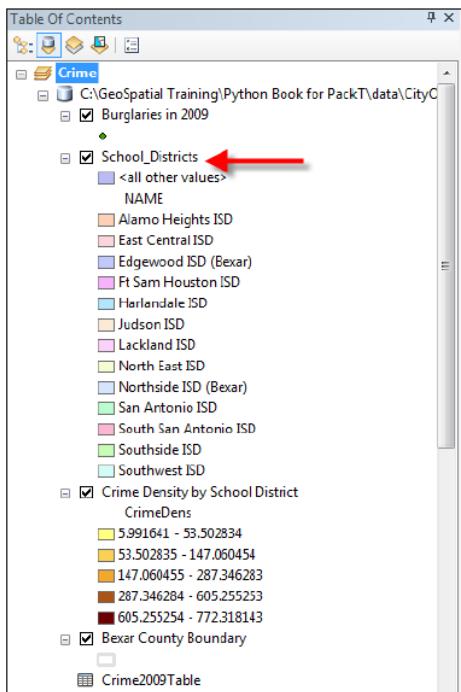
```
layer =
mapping.Layer(r"C:\PythonData\data\School_Districts.lyr")
```

4. เพิ่มเลเยอร์ลงในเฟรมข้อมูล Crime ด้วยเมธอด AddLayer ดังนี้

```
mapping.AddLayer(df,layer,"AUTO_ARRANGE")
```

AUTO_ARRANGE คือให้โปรแกรมจัดรูปแบบของเลเยอร์แบบอัตโนมัติ

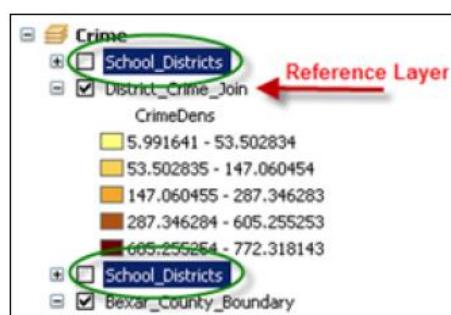
5. เมื่อสั้นรันสคริปต์ โปรแกรมจะเพิ่มแฟ้มเลเยอร์ชื่อ School_Districts.lyr ลงในเฟรมข้อมูลชื่อ Crime ดังรูปที่ 16.8



รูปที่ 16.8 แสดงการเพิ่มเลเยอร์ชื่อ School_Districts ลงในเฟรมข้อมูล Crime

3.13 การเพิ่มเลเยอร์ลงในแผนที่โดยการกำหนดตำแหน่งในเฟรมข้อมูล

เมื่อใช้ AddLayer() กำหนดที่เพิ่มเลเยอร์ลงในเฟรมข้อมูล แต่ไม่สามารถกำหนดตำแหน่งในการเพิ่มเลเยอร์ได้ ในหัวข้อนี้จะใช้เมธอดชื่อ InsertLayer เพื่อสามารถกำหนดตำแหน่งการเพิ่มเลเยอร์ได้ จากตัวอย่างถ้าต้องการเพิ่มเลเยอร์ชื่อ School_Districts ดังรูปที่ 16.9 จะต้องเลือกเลเยอร์ที่ใช้สำหรับอ้างอิงก่อน ในตัวอย่างนี้จะใช้เลเยอร์ชื่อ District_Crime_Join เป็นเลเยอร์อ้างอิง ผู้เขียนโปรแกรมสามารถเพิ่มเลเยอร์ School_Districts ในลำดับก่อนหรือหลังเลเยอร์ที่อ้างอิงได้ โดยมีขั้นตอนการเพิ่มเลเยอร์ดังนี้



รูปที่ 16.9 เพิ่มเลเยอร์ชื่อ School_Districts ก่อนหรือหลังเลเยอร์ที่อ้างอิง (District_Crime_Join)

1. เปิดแฟ้มชื่อ Crime2.mxd และทำการโหลดโมดูล mapping พร้อมกับกำหนดให้ใช้แผนที่ปัจจุบันที่โปรแกรม ArcMap กำลังเปิดใช้งานอยู่

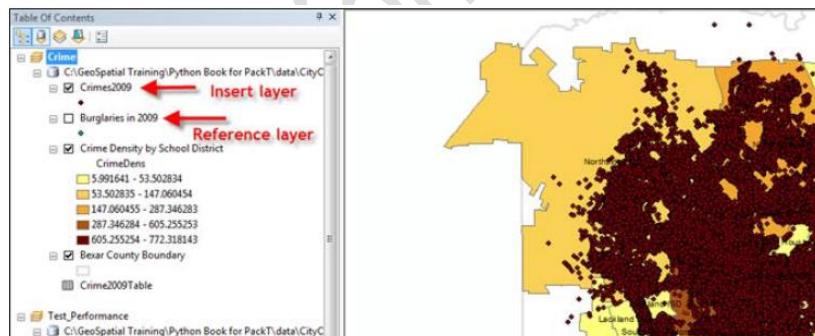

```
import arcpy.mapping as mapping
mxd = mapping.MapDocument("CURRENT")
```
2. ทำการอ้างอิงไปยังเฟรมข้อมูลที่ต้องการเพิ่มเลเยอร์ ในที่นี่คือเฟรมข้อมูลชื่อ Crime


```
df = mapping.ListDataFrames(mxd, "Crime")[0]
```
3. กำหนดเลเยอร์ที่ใช้เพื่ออ้างอิงชื่อ burglaries* (* คือตัวอักษรหรือตัวเลขใดๆ ก็ได้) ในเฟรมข้อมูล Crime


```
refLayer = mapping.ListLayers(mxd, "Burglaries*", df)[0]
```
4. กำหนดเลเยอร์ที่ต้องการเพิ่มชื่อ Crime2009 ซึ่งเก็บอยู่ในไดเรกทรอรี C:\PythonData\data\CityOfSanAntonio.gdb ซึ่งเป็นฐานข้อมูลอาชญากรรมในเมือง SanAntonio


```
insertLayer =
mapping.Layer(r"C:\PythonData\data\CityOfSanAntonio.
gdb\Crimes2009")
```
5. เพิ่มเลเยอร์ที่เลือกไปยังเฟรมข้อมูลชื่อ Crime โดยเพิ่มให้อยู่ในลำดับก่อนหน้าเลเยอร์ชื่อ Burglaries


```
mapping.InsertLayer(df, refLayer, insertLayer, "BEFORE")
```
6. สั่งรันสคริปต์ ผลลัพธ์ที่ได้แสดงในรูปที่ 16.10



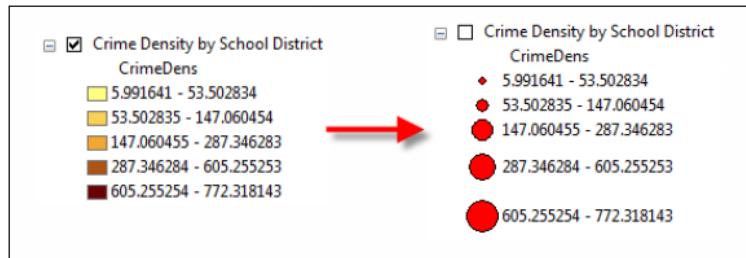
รูปที่ 16.10 เพิ่มเลเยอร์ชื่อ Crime2009 ในลำดับก่อนเลเยอร์อ้างอิง Burglaries in 2009

นอกเหนือจากการเมธอด InsertLayer แล้วผู้เขียนโปรแกรมสามารถเรียกใช้เมธอด MoveLayer ในการเปลี่ยนตำแหน่งของเลเยอร์ได้ตามต้องการ แต่มีข้อยกเว้นว่าต้องอยู่ภายใต้เฟรมข้อมูลเดียวกันเท่านั้น

3.14 การเปลี่ยนสัญลักษณ์ (Symbology) ในเลเยอร์

โมดูล arcpy.mapping ได้เตรียมเมธอดที่ใช้สำหรับแก้ไขหรือปรับปรุงสัญลักษณ์ต่างๆ ที่ใช้แสดงในเลเยอร์ไว้ชื่อ UpdateLayer() ตัวอย่างเช่น เมื่อผู้เขียนโปรแกรมต้องการปรับปรุง

สัญลักษณ์เดิมในレイเยอร์ Crime Density by School District จากสีเหลี่ยมเป็นวงกลม แสดงในรูปที่ 16.11 สามารถทำได้ดังนี้



รูปที่ 16.11 ปรับปรุงสัญลักษณ์สี่เหลี่ยมไปเป็นวงกลมใน Crime Density by School District เลเยอร์

1. เปิดแฟ้มชื่อ Crime2.mxd และทำการโหลดโมดูล mapping พร้อมกับกำหนดให้ใช้แผนที่ปัจจุบันที่โปรแกรม ArcMap กำลังเปิดใช้งานอยู่

```
import arcpy.mapping as mapping
mxd = mapping.MapDocument("CURRENT")
```
2. ทำการอ้างอิงไปยังเฟรมข้อมูลที่ต้องการเพิ่มレイเยอร์ ในที่นี่คือเฟรมข้อมูลชื่อ Crime

```
df = mapping.ListDataFrames(mxd, "Crime") [0]
```
3. กำหนดชื่อของレイเยอร์ที่ต้องการปรับปรุงสัญลักษณ์ ในที่นี่คือ เลเยอร์ชื่อ Crime Density by School District

```
updateLayer = mapping.ListLayers(mxd, "Crime Density by School District", df) [0]
```

ผลจากการเรียกเมธอดนี้ จะได้ค่าที่ส่งกลับมาคือ อ็อปเจกต์ของレイเยอร์ Crime Density by School District เพียงค่าเดียวเท่านั้น
4. กำหนดレイเยอร์ที่ต้องการใช้ปรับปรุงชื่อ CrimeDensityGradSym.lyr

```
sourceLayer =
mapping.Layer(r"C:\PythonData\data\CrimeDensityGradSym.lyr")
```
5. เรียกเมธอด UpdateLayer เพื่อปรับปรุงสัญลักษณ์ของレイเยอร์

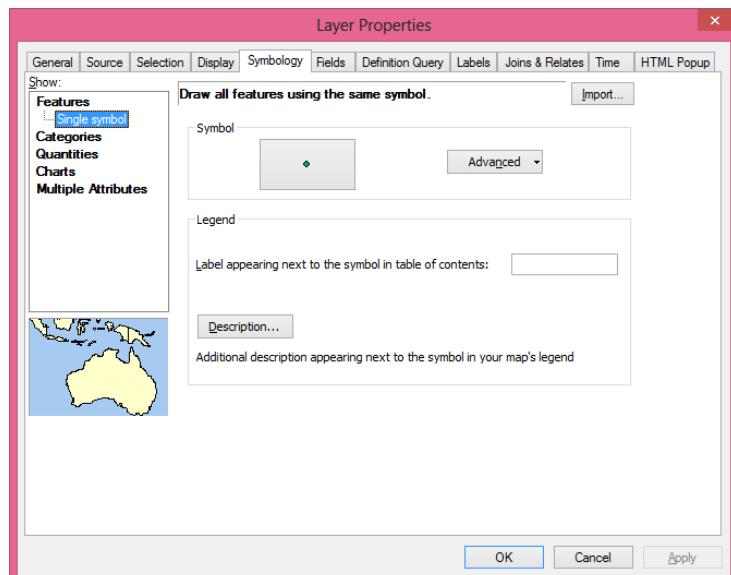
```
mapping.UpdateLayer(df, updateLayer, sourceLayer, True)
```

เมธอด UpdateLayer ต้องการ 4 พารามิเตอร์คือ ตัวแปรอ้างอิงเฟรมข้อมูล (df), อ็อปเจกต์ของレイเยอร์ที่ต้องการปรับปรุง (updateLayer), ตัวแปรอ้างอิงไปยังレイเยอร์ใหม่ (sourceLayer) และ True หมายถึงต้องการปรับปรุงเฉพาะสัญลักษณ์เท่านั้น
6. ส่งรันสคริปต์ และสังเกตุレイเยอร์ชื่อ Crime Density by School District สัญลักษณ์ที่ใช้จะเปลี่ยนจากสี่เหลี่ยมเป็นวงกลม เมธอด UpdateLayer มีความสามารถในการลบレイเยอร์ได้ หรือสามารถใช้เมธอด RemoveLayer() แทนก็ได้

3.15 ปรับปรุงคุณสมบัติต่างๆ ในレイเยอร์ (Layer properties)

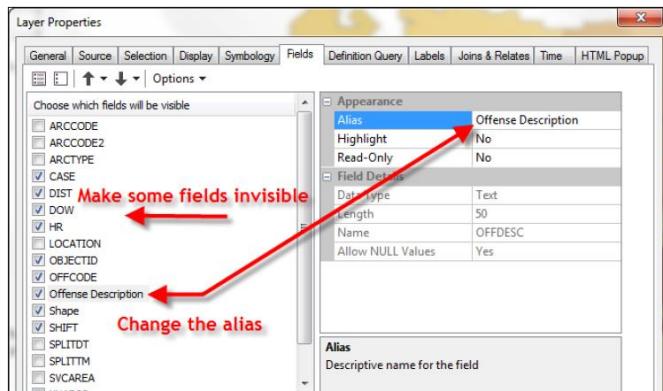
เมธอด UpdateLayer มีความสามารถในการแก้ไขคุณสมบัติต่างๆ ของเลเยอร์ได้ เช่น ชื่อ aliases, symbology, query definitions และ label เป็นต้น ในเบื้องต้นจะแนะนำการใช้ ArcMap ในการแก้ไขคุณสมบัติของเลเยอร์ชื่อ Burglaries in 2009 ต่อจากนั้นจะใช้ Python สรุปต่อในการแก้ไขเลเยอร์ดังกล่าวอีกรอบในภายหลัง

1. เปิดแฟ้มชื่อ Crime2.mxd ด้วยโปรแกรม ArcMap
2. ดับเบิลคลิกเลเยอร์ชื่อ Burglaries in 2009 ซึ่งโปรแกรมจะแสดงหน้าต่างของ Layer properties แสดงในรูปที่ 16.12



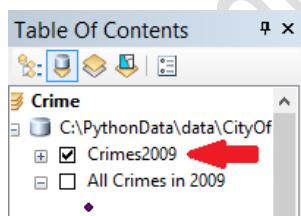
รูปที่ 16.12 แสดง Layer properties

3. คลิกแท็บ General ในส่วน Layer Name: ทดสอบเปลี่ยนชื่อจาก Burglaries in 2009 เป็น Burglaries in 2009 – No Forced Entry
4. คลิกแท็บ Definition Query ให้เพิ่มข้อความว่า "OFFDESC" = 'BURGL-FORCED ENTRY-N' หรือใช้ Query Builder แทนก็ได้
5. คลิกแท็บ Fields คลิกเลือก OFFDESC และเปลี่ยน alias จาก OFFDESC เป็น Offense Description ดังรูปที่ 16.13



รูปที่ 16.13 เปลี่ยนคุณสมบัติ OFFDESC เป็น Offense Description

6. คลิกปุ่ม OK, จากนั้นคลิกขวาที่レイเยอร์ Burglaries – No Forced Entry และบันทึกแฟ้มใหม่เป็นชื่อ BurglariesNoForcedEntry.lyr เก็บไว้ในไดเรกทรอรี C:\PythonData\data
7. คลิกขวาที่ Burglaries – No Forced Entry อีกครั้ง และเลือก Remove
8. จากเมนูหลักใน ArcMap เลือกปุ่ม Add Data เพิ่ม Crimes2009 จากฐานข้อมูล CityOfSanAntonio เข้ามาในแผนที่ Crime2 ดังรูปด้านล่าง



9. เปิดโปรแกรม Python window เพื่อเขียนสคริปต์ โดยเปิดแฟ้มชื่อ Crime2.mxd โหลดโมดูล mapping และกำหนดใช้แผนที่ปัจจุบัน

```
import arcpy.mapping as mapping
mxd = mapping.MapDocument("CURRENT")
```
10. ทำการอ้างอิงไปยังเฟรมข้อมูลชื่อ Crime

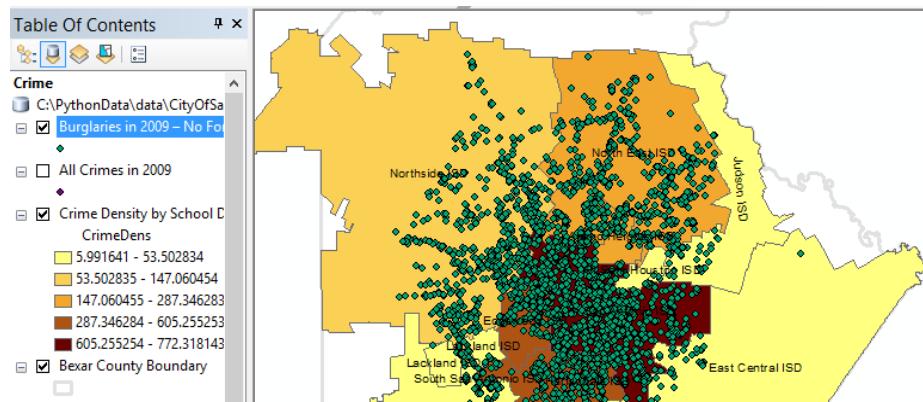
```
df = mapping.ListDataFrames(mxd, "Crime") [0]
```
11. กำหนดชื่อของレイเยอร์ที่ต้องการแก้ไขชื่อ Crimes2009

```
updateLayer = mapping.ListLayers(mxd, "Crimes2009", df) [0]
```
12. เลือกแฟ้มレイเยอร์ที่ต้องการใช้ปรับปรุงชื่อ BurglariesNoForcedEntry.lyr

```
sourceLayer =
mapping.Layer(r"C:\PythonData\data\BurglariesNoForcedEntr
y.lyr")
```
13. ใช้เมธอด UpdateLayer() เพื่อสั่งให้แก้ไขレイเยอร์ใหม่

```
mapping.UpdateLayer(df, updateLayer, sourceLayer, False)
```

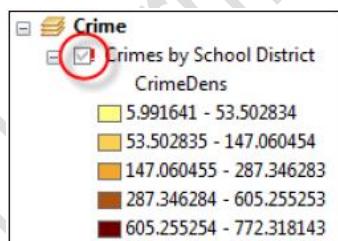
False คือแก้ไขเฉพาะคุณสมบัติต่างๆ เท่านั้น โดยไม่รวมสัญลักษณ์
14. สั่งรันสคริปต์ สังเกตเห็นว่าレイเยอร์ชื่อว่า Crimes2009 จะเปลี่ยนเป็นレイเยอร์ Burglaries in 2009 – No Forced Entry ดังรูปที่ 16.14



รูปที่ 16.14 การแก้คุณสมบัติของเลเยอร์จาก Crimes2009 เป็น Burglaries in 2009

3.16 การค้นหาตำแหน่งข้อมูลต้นฉบับของแฟ้มแผนที่และแฟ้มเลเยอร์

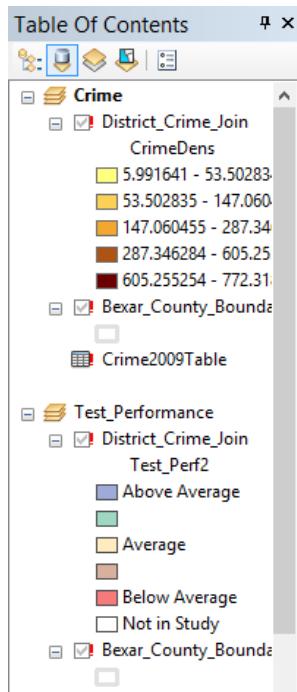
โดยปกติการดำเนินการใดๆ กับแผนที่จำเป็นต้องกำหนดตำแหน่งเพื่อนำเข้าและจัดเก็บแฟ้มข้อมูลก่อนเสมอ (ต้องทำเป็นอันดับแรกในการใช้งานแผนที่) เมื่อตำแหน่งที่เก็บของแฟ้มไม่ได้ระบุไว้หรือระบุไม่ถูกต้องจะส่งผลให้ไม่สามารถแสดงข้อมูลของแผนที่ได้ ดังตัวอย่างในรูปด้านล่าง



จากรูปแสดงให้เห็นว่าไม่ได้กำหนดแหล่งข้อมูลไว้กับแผนที่ชื่อ Crime ไว้ ดังนั้นโปรแกรมจะแสดงเป็นเครื่องหมาย แต่เป็นสีเทาให้ผู้ใช้งานได้เห็น ในหัวข้อนี้จะใช้เมธอดชื่อว่า ListBrokenDataSources() เพื่อค้นหาว่าแหล่งข้อมูลที่เสียหายอยู่ที่ใดบ้าง ดังนี้

1. เปิดแฟ้มชื่อ Crime_BrokenDataLinks.mxd ด้วยโปรแกรม ArcMap ในไดเรคทรอรี

C:\PythonData\Ch16 ดังรูปที่ 16.15



รูปที่ 16.15 แสดงการกำหนดแหล่งข้อมูลผิดพลาด

จากรูปที่ 16.15 แสดงให้เห็นว่าเฟรมข้อมูลและเลเยอร์ทั้งหมดในแฟ้มแผนที่ Crime ไม่สามารถเปิดใช้งานได้ โดยแสดงเป็น สีเทา

15. ปิดโปรแกรม ArcMap และเปิดโปรแกรม Python IDLE ภายนอกโปรแกรม ArcMap
โหลดโมดูล mapping

```
import arcpy.mapping as mapping
```

2. กำหนดให้ไฟล์เรียกใช้แผนที่ชื่อ Crime_BrokenDataLinks.mxd โดยตรงจาก

สาร์ดิสก์ โดยเก็บอยู่ในไดเรคทรอรี C:\PythonData\Ch16

```
mxds =  
mapping.MapDocument(r"c:\PythonData\Ch16\Crime_BrokenData  
Links.mxd")
```

3. เรียกใช้เมธอด ListBrokenDataSources() เพื่อดึงแหล่งข้อมูลที่เกิดความผิดพลาดของ
แผนที่ Crime_BrokenDataLinks

```
lstBrokenDS = mapping.ListBrokenDataSources(mxd)
```

4. ข้อมูลที่ได้จากการเมธอด ListBrokenDataSources() คือรายการของเฟรมข้อมูลที่ไม่
สามารถเชื่อมโยงไปยังแหล่งข้อมูลได ผู้เขียนโปรแกรมจำเป็นต้องใช้ for เพื่อช่วยใน
การแสดงผลข้อมูลดังกล่าว ดังนี้

```
for layer in lstBrokenDS:  
    print layer.name
```

5. บันทึกแฟ้มข้อมูลเป็นชื่อ FindFixBrokenData.py ในไดเรคทรอรี C:\PythonData\Ch16

สั่งรันโปรแกรม ผลลัพธ์ที่ได้คือ



```
District_Crime_Join
Bexar_County_Boundary
District_Crime_Join
Bexar_County_Boundary
Bexar_County_Boundary
Texas_Counties_LowRes
School_Districts
Crime_surf
Bexar_County_Boundary
Crime2009Table
```

จากผลลัพธ์ด้านบนแสดงรายการเฟรมข้อมูลที่ไม่สามารถเชื่อมโยงไปยังแหล่งข้อมูลของแฟ้มได้ ผู้เขียนโปรแกรมสามารถแสดงรายการเลเยอร์ที่มีแหล่งข้อมูลผิดพลาดได้ เช่นกัน โดยเปลี่ยนจากแฟ้มที่ มีนามสกุลเป็น .mxd เป็น .lyr

3.17 การแก้ปัญหาแหล่งข้อมูลผิดพลาดด้วยเมธอด findAndReplaceWorkspacePaths

แหล่งเก็บข้อมูลที่ใช้กับแผนที่หรือเรียกว่า workspace เป็นตำแหน่งที่ใช้เก็บข้อมูลต่างๆ ที่ จำเป็นสำหรับแผนที่ เช่น shapefiles, personal geodatabase, file geodatabase หรือ ArcSDE connection เป็นต้น เมธอด findAndReplaceWorkspacePaths() ในโมดูล arcpy.mapping สามารถ แก้ไขแหล่งข้อมูลที่ผิดพลาดได้ใน 3 ประเภทคือ MapDocument, Layer และ TableView

1. เปิดแฟ้มชื่อ Crime_BrokenDataLinks.mxd ด้วยโปรแกรม ArcMap ในไดเรคทรอรี C:\PythonData\Ch16
2. คลิกขวาที่เลเยอร์ไดเลเยอร์หนึ่งแล้วเลือก Properties
3. คลิกเลือกที่แท็บ Source \Rightarrow ในส่วนของ Data Source \Rightarrow Database จะชี้ไปยัง ไดเรคทรอรี C:\PythonData\Ch16\Data\OldData\CityOfSanAntonio.gdb ซึ่งไม่ ถูกต้อง เนื่องจากแฟ้มจริงอยู่ในไดเรคทรอรี C:\PythonData\data
4. เปิดโปรแกรม Python IDLE และโหลดโมดูล mapping
`import arcpy.mapping as mapping`
6. กำหนดให้ Python เรียกใช้แผนที่ชื่อ Crime_BrokenDataLinks.mxd โดยตรงจาก สาร์ดิสก์ โดยเก็บอยู่ในไดเรคทรอรี C:\PythonData\Ch16
`mxds =`
`mapping.MapDocument(r"c:\PythonData\Ch16\Crime_BrokenDataLinks.mxd")`
7. ใช้เมธอด findAndReplaceWorkspacePaths() เพื่อแก้ไขที่อยู่ของไดเรคทรอรีที่เป็น แหล่งข้อมูล (Data Source) ให้ถูกต้อง
`mxds.findAndReplaceWorkspacePaths(r"`
`C:\PythonData\Ch16\Data\OldData\CityOfSanAntonio.gdb",`
`r"C:\PythonData\data\CityOfSanAntonio.gdb")`

8. เรียกใช้เมธอด SaveACopy() เพื่อบันทึกเป็นแฟ้มใหม่ชื่อ

Crime_DataLinksFixed.mxd

```
mxd.saveACopy(r"C:\PythonData\Ch16\Crime_DataLinksFixed.mxd")
```

9. บันทึกสคริปต์ในไฟล์ C:\Python\Ch16\MapDocumentFindReplace.py

10. สร้างสคริปต์ นำเข้าไปในไฟล์ Python ที่ต้องการ เช่น

C:\PythonData\Ch16\Crime_DataLinksFixed.mxd และคลิกขวาที่เลเยอร์ใดเลเยอร์หนึ่งแล้วเลือก Properties → คลิกแท็บ Source → Data Source → Database: สังเกตว่าได้ระบุชื่อรีสурсแก้ไขเป็น C:\PythonData\data\CityOfSanAntonio.gdb



Tips: ตัวอักษร r ที่อยู่หน้าข้อความที่ระบุถึงไฟล์เดิมที่ต้องการ เช่น

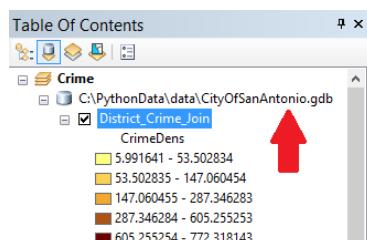
(r"C:\PythonData\Ch16\Crime_DataLinksFixed.mxd") คือบอกให้ไฟล์ Python ไม่ตัดตอนตัวอักษรใดๆ ในสตริงออก เช่น ถ้าใช้คำสั่ง print "C:\\test" ผลลัพธ์ที่ได้คือ "C:\\test" แต่ถ้าใช้ r ร่วมกับคำสั่งดังกล่าว เช่น print r"C:\\test" หรือ print (r"C:\\test") ผลลัพธ์ที่ได้คือ "C:\\test" เนื่องจากว่าข้อความจะไม่ถูกตัดตอนตัวอักษรใดๆ ออก

เมธอด findAndReplaceWorkspacePaths() สามารถค้นหาและแทนที่แหล่งข้อมูลของเลเยอร์ (Layer) และตาราง (TableView) ได้เช่นเดียวกับ MapDocument

3.18 การแก้ไขแหล่งข้อมูลด้วยคำสั่ง MapDocument.replaceWorkspaces()

เมธอด replaceWorkspaces() ทำงานเหมือนกับเมธอด findAndReplaceWorkspacePaths() แตกต่างตรงที่เมธอด replaceWorkspaces() สามารถสลับการทำงานจาก workspace หนึ่งไปยัง workspace อื่นๆ ได้ เช่น ขณะทำงานอยู่กับ file geodatabase สามารถสลับไปทำงานกับ personal geodatabase ได้ แต่อย่างไรก็ตาม เมธอดดังกล่าวจะทำงานกับ workspace ใด workspace หนึ่ง ก็เวลาใดเวลาหนึ่งเท่านั้น วิธีการใช้งานดังนี้

1. เปิดแฟ้ม C:\PythonData\Ch16\Crime_DataLinksFixed.mxd ด้วยโปรแกรม ArcMap
2. โปรดจำไว้ว่าข้อมูลที่ใช้สร้างเลเยอร์และตารางทั้งหมดเก็บอยู่ใน file geodatabase ชื่อ CityOfSanAntonio.gdb ดังรูปข้างล่าง



3. เปิดโปรแกรมไพธอน IDLE และโหลดโมดูล mapping

```
import arcpy.mapping as mapping
```

4. กำหนดแผนที่ชื่อ Crime_BrokenDataLinks.mxd โดยตรงจากอาร์ดิสก์

```
mx = mapping.MapDocument(r"c:\PythonData\Ch16\Crime_BrokenDataLinks.mxd")
```

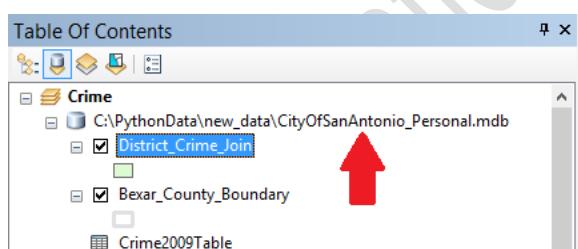
5. เรียกใช้เมธอด replaceWorkspaces() เพื่อแก้ไขแหล่งข้อมูลเก่าเป็นแหล่งข้อมูลใหม่
`mx.replaceWorkspaces(r"c:\PythonData\data\CityOfSanAntonio.gdb", "FILEGDB_WORKSPACE", r"C:\PythonData\new_data\CityOfSanAntonio_Personal.mdb", "ACCESS_WORKSPACE")`

6. บันทึกเป็นแฟ้มแผนที่ใหม่ชื่อ Crime_DataLinksUpdated.mxd

```
mx.saveACopy(r"C:\PythonData\Ch16\Crime_DataLinksUpdated.mxd")
```

7. บันทึกสคริปต์ไฟล์เป็น C:\Python\Ch16\MapDocumentReplaceWorkspace.py

8. สั่งรันสคริปต์ และสังเกตว่าฐานข้อมูลเดิมเปลี่ยนเป็นฐานข้อมูลใหม่หรือไม่ ดังรูป
ด้านล่าง



สำหรับเมธอด replaceWorkspaces() จะมีคีย์เวิร์ด 2 ตัวคือ FILEGDB_WORKSPACE และ ACCESS_WORKSPACE ซึ่งหมายถึงชนิดของฐานข้อมูลที่ใช้งาน สำหรับฐานข้อมูลอื่นๆ แสดงดังในตารางด้านล่าง

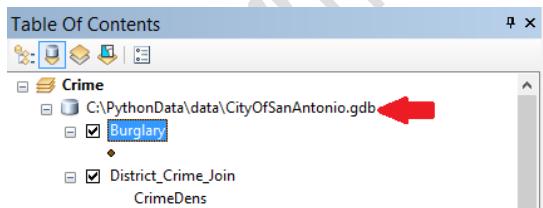
คีย์เวิร์ด	ฐานข้อมูล
ACCESS_WORKSPACE	A personal geodatabase or Access workspace
ARCINFO_WORKSPACE	An ArcInfo coverage workspace
CAD_WORKSPACE	A CAD file workspace
EXCEL_WORKSPACE	An Excel file workspace
FILEGDB_WORKSPACE	A file geodatabase workspace
NONE	Used to skip the parameter
OLEDB_WORKSPACE	An OLE database workspace
PCCOVERAGE_WORKSPACE	A PC ARC/INFO Coverage workspace
RASTER_WORKSPACE	A raster workspace

SDE_WORKSPACE	An SDE geodatabase workspace
SHAPEFILE_WORKSPACE	A shapefile workspace
TEXT_WORKSPACE	A text file workspace
TIN_WORKSPACE	A TIN workspace
VPF_WORKSPACE	A VPF workspace

3.19 แก้ไขแหล่งข้อมูลในระดับเลเยอร์ (Layer) และตาราง (TableView)

ในหัวข้อที่แล้วได้กล่าวถึงการแก้ไขการเชื่อมโยงแหล่งข้อมูลกับ MapDocument ด้วยเมธอด replaceWorkspaces() แต่สำหรับการแก้ไขการเชื่อมโยงแหล่งข้อมูลสำหรับเลเยอร์และตารางจะใช้เมธอด replaceDataSource() แทน ดังนี้

1. เปิดแฟ้ม C:\PythonData\Ch16\Crime_DataLinksLayer.mxd ด้วยโปรแกรม ArcMap ในแผนที่ดังกล่าวจะมีเลเยอร์ชื่อ Burglary ซึ่งเชื่อมโยงกับแฟ้มฐานข้อมูล geodatabase ชื่อ CityOfSanAntonio.gdb ดังรูปด้านล่าง เมื่อต้องการแก้ไขแหล่งข้อมูลใหม่ให้กับเลเยอร์ Burglary สามารถทำได้ดังนี้



2. เปิดโปรแกรม Python IDLE และโหลดโมดูล mapping
import arcpy.mapping as mapping
3. กำหนดแหนณที่ชื่อ Crime_DataLinksLayer โดยตรงจากชาร์ดดิสก์
4. สร้างการเชื่อมโยงไปยังเฟรมข้อมูล Crime
df = mapping.ListDataFrames(mxd, "Crime") [0]
5. สร้างการเชื่อมโยงไปยังเลเยอร์ Burglary
lyr = mapping.ListLayers(mxd, "Burglary", df) [0]
6. เรียกใช้เมธอด replaceDataSource() เพื่อแก้ไขแหล่งข้อมูลเก่าเป็นแหล่งข้อมูลใหม่ โดยเปลี่ยนจาก C:\PythonData\data\CityOfSanAntonio.gdb เป็น C:\PythonData\data\Burglaries_2009.shp
lyr.replaceDataSource(r"C:\PythonData\data", "SHAPEFILE_WORKSPACE", "Burglaries_2009")

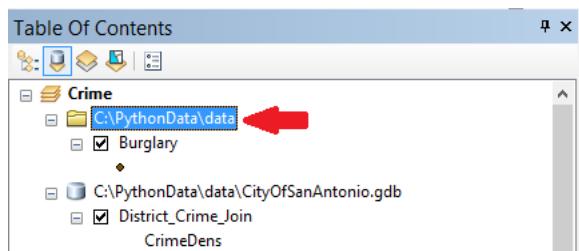
คีย์เวิร์ด SHAPEFILE_WORKSPACE เพื่อใช้ระบุว่าแหล่งข้อมูลใหม่เป็นแฟ้มชนิด shapefile ชื่อว่า Burglaries_2009.shp

- บันทึกแฟ้มแผนที่เป็นชื่อใหม่คือ Crime_DataLinksNewLayer.mxd

```
mxd.saveACopy(r"C:\PythonData\Ch16\Crime_DataLinksNewLayer.mxd")
```

- บันทึกสคริปต์พร้อมเป็น C:\Python\Ch16\LayerReplaceDataSource.py

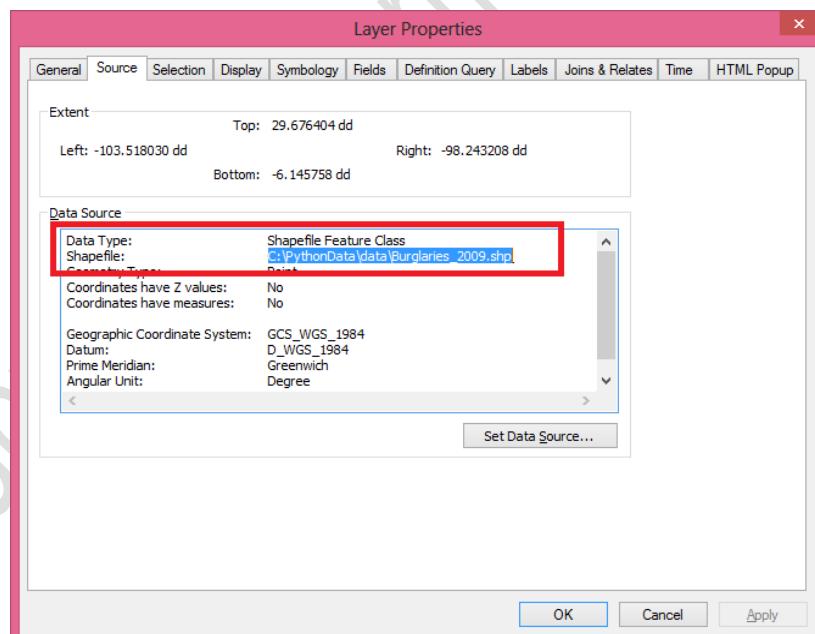
- สั่งรันสคริปต์ และสังเกตว่าฐานข้อมูลเดิมเปลี่ยนเป็นฐานข้อมูลใหม่หรือไม่ ดังรูปด้านล่าง



- คลิกขวาที่เลเยอร์ Burglary แล้วเลือก Properties

- คลิกแท็ป Source ในส่วนของ Shapefile จะซึ่งไปยังแฟ้มชื่อ

C:\PythonData\data\Burglaries_2009.shp ดังรูปที่ 16.16



รูปที่ 16.16 แสดงการเปลี่ยนแหล่งข้อมูลด้วยเมธอด replaceDataSource()

จากรูปสังเกตเห็นว่าเลเยอร์ Burglary ซึ่งไปยังแฟ้มข้อมูล Burglaries_2009.shp แต่เลเยอร์อื่นๆ ที่อยู่ในแผนที่เอกสารเดียวกันจะยังคงเหมือนเดิม จะเปลี่ยนแปลงเฉพาะเลเยอร์ Burglaries เท่านั้น

3.20 การค้นหาการเชื่อมโยงแหล่งข้อมูลที่ผิดพลาดทั้งหมดในไฟล์เครตรอรี

1. เปิดโปรแกรมไฟรอน IDLE และทำการ import arcpy.mapping และ os package

```
import arcpy.mapping as mapping, os
```
2. กำหนดชื่อไฟล์เครตรอรีที่ต้องการค้นหา ในกรณีนี้จะเริ่มต้นค้นหาตั้งแต่ไฟล์ C: และทุกๆ ไฟล์เครตรอรีอยู่ที่อยู่ภายใต้ไฟล์ดังกล่าวทั้งหมด

```
path = r"C:"
```
3. เปิดแฟ้มชื่อ "BrokenDataList.txt" เพื่อใช้สำหรับเก็บชื่อเลเยอร์ที่เชื่อมโยงแหล่งข้อมูลผิดพลาด

```
f = open('BrokenDataList.txt', 'w')
```
4. เรียกใช้เมธอด os.walk() ร่วมกับ for เพื่อแสดงรายชื่อไฟล์เครตรอรีทั้งหมดภายในไฟล์ C: มาแสดง

```
for root, dirs, files in os.walk(path):
```
5. ภายในลูป for ให้ทำการสร้างลูป for อีกชั้นอยู่ภายใต้ไฟล์เครตรอรีทั้งหมดที่อยู่ในตัวแปร files ออกแบบโดยทำงานร่วมกับเมธอด os.path.splitext() ซึ่งจะแยกชื่อแฟ้มพร้อมกับส่วนขยาย (นามสกุล) ออกแบบจากตัวแปร files

```
for filename in files:
    basename, extension = os.path.splitext(filename)
```
6. ถ้าแฟ้มมีนามสกุล .mxd โปรแกรมจะสร้างการเชื่อมโยงไปยังแฟ้มดังกล่าวด้วยเมธอด mapping.MapDocument(fullPath) จากนั้นจะเขียนข้อความว่า "MXD: " + filename + "\n" ลงแฟ้ม "BrokenDataList.txt" ต่อจากนั้นทำการตรวจสอบการเชื่อมโยงว่าผิดพลาดหรือไม่ด้วยเมธอด mapping.ListBrokenDataSources(mxd) ถ้าการเชื่อมโยงผิดพลาดโปรแกรมจะเขียนชื่อเลเยอร์ลงแฟ้ม "BrokenDataList.txt" ด้วยเมธอด f.write("\t" + brknItem.name + "\n")

```
if extension == ".mxd":
    fullPath = os.path.join(path, filename)
    mxd = mapping.MapDocument(fullPath)
    f.write("MXD: " + filename + "\n")
    brknList = mapping.ListBrokenDataSources(mxd)
    for brknItem in brknList:
        f.write("\t" + brknItem.name + "\n")
```
7. ปิดแฟ้มข้อมูล "BrokenDataList.txt"

```
f.close()
```
8. สรุปคำสั่งทั้งหมดแสดงได้ดังนี้

```
import arcpy.mapping as mapping, os
path = r"C:"
f = open('BrokenDataList.txt', 'w')
for root, dirs, files in os.walk(path):
    for filename in files:
        basename, extension = os.path.splitext(filename)
```

```
if extension == ".mxd":  
    fullPath = os.path.join(path,filename)  
    mxd = mapping.MapDocument(fullPath)  
    f.write("MXD: " + filename + "\n")  
    brknList = mapping.ListBrokenDataSources(mxd)  
    for brknItem in brknList:  
        f.write("\t" + brknItem.name + "\n")  
f.close()
```

9. บันทึกสคริปต์ชื่อ C:\Python\Ch16\FindAllBrokenDataSources.py
10. สร้างสคริปต์ และเปิดแฟ้มชื่อ BrokenDataList.txt เพื่อดูผลลัพธ์ของโปรแกรม



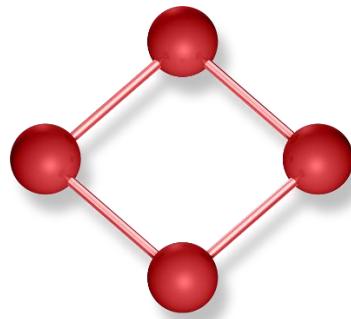
OUTPUT

```
MXD: Crime1.mxd  
MXD: Crime2.mxd  
MXD: Crime_BrokenDataLinks.mxd  
    District_Crime_Join  
    Bexar_County_Boundary  
    District_Crime_Join  
    Bexar_County_Boundary  
    Bexar_County_Boundary  
    Texas_Counties_LowRes  
    School_Districts  
    Crime_surf  
    Bexar_County_Boundary  
    Crime2009Table  
...
```

ในการเขียนโปรแกรมไพธอนร่วมกับ ArcGIS มีรายละเอียดอีกมากมาย ในบทนี้จะเป็นเพียงการแนะนำการเขียนโปรแกรมเบื้องต้นเท่านั้น ถ้าผู้อ่านต้องการรายละเอียดเพิ่มเติมสามารถเปิดอ่านได้โดยตรงจาก help ของ ArcGIS ซึ่งจะมีทั้งคำอธิบายและตัวอย่างโปรแกรมไพธอนให้ด้วย

บทที่ 17

โครงสร้างข้อมูลเบื้องต้น (Basic Data Structures)



อัลกอริทึม (Algorithm) เป็นวิธีการดำเนินงานของโปรแกรมบนระบบคอมพิวเตอร์ เพื่อให้สามารถแก้ปัญหาได้มีประสิทธิภาพสูงสุด (Optimal) บนทรัพยากรที่มีอย่างจำกัด ซึ่งทรัพยากรดังกล่าวมี 2 ประเภทคือ เวลาที่ใช้ในการประมวลผล (Processing Time) และพื้นที่หน่วยความจำ (Memory) ที่ใช้งาน ถ้าโปรแกรมตั้งแต่ 2 โปรแกรมขึ้นไป ดำเนินการแก้ปัญหาชนิดเดียวกันกับข้อมูลชุดเดียวกันแล้ว โปรแกรมที่ใช้ทรัพยากรทั้ง 2 ชนิดน้อยที่สุดแสดงว่า อัลกอริทึมของโปรแกรมนั้นๆ มีประสิทธิภาพที่สูงกว่า นั่นเอง ในบทนี้จะกล่าวถึงวิธีที่ใช้วัดประสิทธิภาพของอัลกอริทึมในเบื้องต้นก่อน จากนั้นจะนำเอาวิธีการดังกล่าวไปประยุกต์ใช้กับอัลกอริทึมการจัดเรียง (Sorting) และการค้นหาข้อมูล (Searching) ต่อไป

1. การประเมินประสิทธิภาพอัลกอริทึมจากเวลาที่ใช้ทำงาน (Run time)

วิธีการแรกที่ใช้วัดประสิทธิภาพอัลกอริทึมคือ วัดจากการระยะเวลาการทำงานตั้งแต่เริ่มต้นจนจบ โดยใช้สัญญาณนาฬิกาจริงจากเครื่องคอมพิวเตอร์ (Computer's clock) เรียกวิธีการนี้ว่า benchmarking หรือ profiling โดยออกแบบให้ชุดของข้อมูลที่ใช้สำหรับทดสอบมีความหลากหลาย เช่น ขนาดข้อมูลที่แตกต่างกัน และชนิดข้อมูลไม่เหมือนกันเป็นต้น ทำการทดสอบกับชุดข้อมูลดังกล่าวหลาย ๆ ครั้งแล้วจึงหาค่าเฉลี่ย จากตัวอย่าง ถ้าพิจารณาอัลกอริทึมการนับเลขจำนวนเต็มตั้งแต่ 1 – N โดย N คือขนาดของปัญหาเป็นเลขจำนวนเต็มที่ไม่เท่ากับ 0 และไม่เป็นค่าลบ ($\sum_{i=1}^n i$) กำหนดให้โปรแกรมทำงาน 5 รอบ โดยในแต่ละรอบ (Epoch) โปรแกรมจะเพิ่มขนาดของปัญหา (N) และจับเวลาเริ่มต้นและจบด้วยเวลาสิ้นสุด ผลลัพธ์ที่ได้คือ เวลาในการทำงานของอัลกอริทึมนั้นเอง ดังตัวอย่างโปรแกรมที่ 17.1

Program Example 17.1: Counting Runtime	
---	--

⇒1	import time
2	
⇒3	problemSize = 10000000

```

4   print("%3s%14s%15s" %("Epoch", "Problem Size", "Seconds"))
5   for count in range(5):
6       start = time.time()
7       work = 0
8       #Start of the algorithm
9       for x in range(problemSize):
10          work += x
11       #Stop of the algorithm
12       elapsed = time.time() - start
13       print("%5d%14d%15f" %(count+1, problemSize, elapsed))
14       problemSize *= 2

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



	Epoch	Problem Size	Seconds
	1	10000000	2.220476
	2	20000000	4.396939
	3	40000000	9.169125
	4	80000000	18.255202
	5	160000000	37.282913

จากตัวอย่างที่ 17.1 บรรทัดที่ 1 โปรแกรมนำเข้าโมดูล time เพื่อใช้สำหรับจับเวลาของโปรแกรมที่ประมวลผล บรรทัดที่ 3 กำหนดขนาดของปัญหา (problemSize) ให้มีขนาดใหญ่เท่ากับ 10,000,000 บรรทัดที่ 5 กำหนดจำนวนรอบในการทำงานให้กับโปรแกรม โดยใช้คำสั่ง for ในที่นี้โปรแกรมจะทำงานทั้งหมด 5 รอบ บรรทัดที่ 6 ทำการเรียกใช้เมธอด time() ซึ่งให้ผลลัพธ์มีหน่วยเป็นวินาที เก็บไว้ในตัวแปร start เพื่อทำหน้าที่เก็บเวลาเริ่มต้นก่อนอัลกอริทึมจะทำงาน บรรทัดที่ 9 และ 10 โปรแกรมจะทำอัลกอริทึมหาร่วมจาก 0 ถึง problemSize -1 โดยเก็บผลลัพธ์สะสมไว้ในตัวแปร work เมื่อโปรแกรมทำงานเสร็จในรอบที่ 1 จะคำนวณหาเวลาเริ่มต้น – เวลาสิ้นสุดของอัลกอริทึม (บรรทัดที่ 12) และพิมพ์ผลลัพธ์ดังกล่าวออกจอภาพ (บรรทัดที่ 13) เมื่อพิมพ์ผลลัพธ์แล้ว โปรแกรมจะเพิ่มขนาดของปัญหาเป็น 2 เท่าหรือ $2N$ (โดยการคูณด้วย 2 ในบรรทัดที่ 14) ส่งผลให้เวลาประมวลผลของอัลกอริทึมเพิ่มขึ้นประมาณ 2 เท่าด้วย ดังแสดงในตัวอย่าง OUTPUT ด้านบน

จากตัวอย่างข้างบนเป็นอัลกอริทึมที่ใช้หาผลรวมของจำนวนนับตั้งแต่ 1 – N (problemSize) โดยอาศัยการทำงานของ for แบบชั้นเดียว แต่ถ้าอัลกอริทึมมีลักษณะในตัวอย่างต่อไปนี้ เวลาที่ใช้ประมวลผลจะมีค่าเป็นอย่างไร

```

for i in range(problemSize):
    for j in range(problemSise):
        work += j

```

คำตอบคือ อาจจะต้องใช้ตลอดทั้งคืนก็เป็นได้ เนื่องจากปัญหามีขนาดเพิ่มขึ้นเป็นการยกกำลังสอง (N^2) นั่นเอง สำหรับวิธีการวัดประสิทธิภาพอัลกอริทึมด้วยการจับเวลา มีข้อจำกัด 2 ประการคือ

- อาจรดware ที่มีความแตกต่างกันจะส่งผลโดยตรงกับความเร็วในการวัดประสิทธิภาพ, ระบบปฏิบัติการที่ใช้งานไม่เหมือนกันจะมีผลกระทบเช่นกัน รวมไปถึงภาษาที่ใช้เขียน

โปรแกรมก็มีผลกระทบด้วย เช่น ภาษาซึ่งทำงานเร็วกว่าภาษาไพธอน เป็นต้น ดังนั้น สรุปว่า ความแตกต่างของอาร์ดแวร์และซอฟต์แวร์จะมีผลกระทบโดยตรงกับการประเมินประสิทธิภาพของอัลกอริทึม

- สำหรับในบางกรณี ข้อมูลที่ใช้ทดสอบมีขนาดใหญ่เกินกว่าที่เครื่องคอมพิวเตอร์จะสามารถคำนวณได้ เมื่อไม่สามารถคำนวณได้ก็ส่งผลให้การจับเวลาไม่บรรลุผลเช่นเดียวกัน

โปรดจำไว้ว่า การประเมินอัลกอริทึมโดยการจับเวลาจะใช้ได้กับบางกรณีเท่านั้น ดังนั้นนักคอมพิวเตอร์ทั้งหลายจึงได้พยายามคิดค้นวิธีการที่จะแก้ปัญหาการประเมินประสิทธิภาพอัลกอริทึมโดยไม่ขึ้นกับอาร์ดแวร์และซอฟต์แวร์ ดังต่อไปนี้

2. การประเมินประสิทธิภาพอัลกอริทึมจากการนับจำนวนบรรทัดคำสั่ง (Counting instructions)

วิธีการนับจำนวนคำสั่งถูกคิดค้นขึ้นเพื่อจะแก้ปัญหาวิธีการแบบจับเวลา เนื่องจากวิธีการดังกล่าวจะขึ้นกับอาร์ดแวร์และซอฟต์แวร์ โดยวิธีการนี้จะใช้วิธีการนับจำนวนบรรทัดของโปรแกรม (lines of code) ที่เขียนขึ้น โปรดจำไว้ว่าเป็นการนับจำนวนคำสั่งของโปรแกรมระดับสูง (High level code) ไม่ใช่คำสั่งระดับล่าง (Machine code) โดยหลักการนี้จะให้ความสนใจเป็นพิเศษกับการนับจำนวนบรรทัดในลูปชุดซ้อน หรือ Nested loop หรือการเรียกเกิด (Recursion) ดังตัวอย่างต่อไปนี้

Program Example 17.2: Counting instructions (Iteration)

```

1 import time
2
3 ➔ problemSize = 1000
4 print("%3s%14s%15s" %("Epoch", "Problem Size", "Iterations"))
5 for count in range(5):
6     number = 0
7     work = 0
8     #Start of the algorithm
9     for x in range(problemSize):
10         for y in range(problemSize):
11             number += 1
12             work += x
13     #Stop of the algorithm
14     print("%5d%14d%15d" %(count+1, problemSize, number))
problemSize *= 2
    
```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



Epoch	Problem Size	Iterations
1	1000	1000000
2	2000	4000000
3	4000	16000000
4	8000	64000000

จากตัวอย่างที่ 17.2 แสดงการนับจำนวนบรรทัดของโปรแกรม (lines of code) โดยบรรทัดที่ 3 โปรแกรมกำหนดค่าให้กับตัวแปร problemSize เท่ากับ 1,000 ซึ่งถ้ากำหนดค่าให้กับตัวแปรดังกล่าว ให้ญี่เกินไป จะส่งผลให้คอมพิวเตอร์ประมวลผลนานเกินไป บรรทัดที่ 6 กำหนดตัวแปร number เท่ากับ 0 เพื่อใช้สำหรับนับจำนวนบรรทัดของโปรแกรม บรรทัดที่ 9 และ 10 สร้างลูป for แบบซ้อน โดยวนลูป ซ้ำจาก 0 ถึง problemSize (1,000) บรรทัดที่ 11 โปรแกรมทำการบวกค่าให้กับตัวแปร number ขึ้นทีละ 1 (number += 1 \Rightarrow number = number + 1) ซึ่งหมายถึง โปรแกรมจะนับเป็น 1 บรรทัด โดยไม่สนใจ ว่าภายในลูปจะมีคำสั่งอื่นๆ ที่ทำงานอยู่ก็ตาม เช่นบรรทัดที่ 11 และ 12 สรุปคือ การวนลูป 1 ครั้ง อัลกอริทึมดังกล่าวจะนับเป็น 1 บรรทัด ตัวอย่างเช่น จาก Epoch 4 มีจำนวนของัญหาเท่ากับ 8,000 ส่งผลให้มีจำนวนบรรทัดในการทำงานเท่ากับ 64,000,000 บรรทัด เป็นต้น

ตัวอย่างโปรแกรมที่ 17.3 แสดงตัวอย่างการนับจำนวนบรรทัดของโปรแกรม Fibonacci โดย โปรแกรมดังกล่าวทำงานในลักษณะเวียนเกิด (Recursion) ดังนี้

Program Example 17.3: Counting instructions (Recursion)

```

1 class Counter(object):
2     def __init__(self):
3         self._number = 0
4
5     def increment(self):
6         self._number += 1
7
8     def __str__(self):
9         return str(self._number)
10
11 def Fibonacci(n, counter):
12     counter.increment()
13     if n < 3:
14         return 1
15     else:
16         return Fibonacci(n-1, counter) + Fibonacci(n-2,
counter)
17
18 problemSize = 2
19 print("%12s%15s" %("Problem Size", "Calls"))
20 for i in range(5):
21     counter = Counter()
22     Fibonacci(problemSize, counter)
23     print("%12d%15d" %(problemSize, counter))
24     problemSize *= 2

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



	Problem Size	Calls
	2	1
	4	5
	8	41
	16	1973

จากผลลัพธ์แสดงให้เห็นว่าขนาดของบัญหาเพิ่มขึ้นเป็น 2^n คือ 2, 4, 8, 16 และ 32 ส่งผลให้จำนวนบรรทัดเพิ่มขึ้นอย่างรวดเร็ว วิธีการนี้มีข้อดีคือไม่ขึ้นต่ออาร์ดแวร์และซอฟต์แวร์ และยังเป็นพื้นฐานที่สำคัญในการออกแบบวิธีการประเมินประสิทธิภาพด้วยวิธีทางคณิตศาสตร์ ซึ่งจะกล่าวต่อไป

3. การประเมินประสิทธิภาพอัลกอริทึมจากการคำนวณหน่วยความจำ (Memory)

สำหรับการคำนวณหน่วยความจำแบ่งเป็น 2 ประเภทคือ การคำนวณหน่วยความจำที่เกิดจากการใช้งานจริงในขณะที่โปรแกรมกำลังทำงาน และการคำนวณทางคณิตศาสตร์ ในส่วนแรกจะกล่าวถึง การคำนวณหน่วยความจำจากการใช้งานจริงก่อน ผู้เขียนแนะนำให้ใช้โมดูลชื่อ psutil ซึ่งสามารถดาวน์โหลดและติดตั้งได้จาก URL: <https://pypi.python.org> เมื่อติดตั้งแล้วสามารถเรียกใช้งานกับ Python ได้ทันที จากโปรแกรมตัวอย่างที่ 17.4 แสดงการใช้โมดูล psutil ในการคำนวณขนาดของหน่วยความจำดังนี้

Program Example 17.4: Counting memory

```

⇒1 import psutil
⇒2 import os
3
⇒4 def memory_usage_psutil():
5      # return the memory usage in MB
6      process = psutil.Process(os.getpid())
⇒7      mem = process.get_memory_info()[0] / float(2 ** 20)
8      return mem
9
10     mem = memory_usage_psutil()
11     print(str(mem) + " MB")

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



20.38 MB

OUTPUT

จากตัวอย่างโปรแกรมที่ 17.4 แสดงการคำนวณหน่วยความจำที่โปรแกรมกำลังใช้งาน โดยบรรทัดที่ 1 นำเข้าโมดูล psutil เพื่อใช้สำหรับแสดงข้อมูลทรัพยากร่างๆ ของเครื่องคอมพิวเตอร์ บรรทัดที่ 2 นำเข้าโมดูล os เพื่อใช้ในการเข้าถึงข้อมูลระบบ เช่น หมายเลขโทรศัพท์, ชื่อไดเรกทอรีปัจจุบัน เป็นต้น บรรทัดที่ 4 – 7 สร้างฟังctionชื่อ memory_usage_psutil() ทำหน้าที่คำนวณขนาดของหน่วยความจำค่าที่ส่งคืนกลับจากฟังctionดังกล่าว คือขนาดหน่วยความจำที่โปรแกรมบัญชีกำลังใช้งานอยู่ มีหน่วยเป็นเมกะไบต์ (MB) โมดูล psutil ยังมีความสามารถในการแสดงข้อมูลอื่นๆ ของระบบได้อีกเป็นจำนวนมากมากดังนี้

เมธอด	คำอธิบาย
psutil.cpu_times()	แสดงข้อมูลเกี่ยวกับหน่วยประมวลผลกลาง (ซีพียู)
psutil.virtual_memory()	แสดงข้อมูลเกี่ยวกับหน่วยความจำชนิดเวอร์ชัล
psutil.disk_partitions()	แสดงข้อมูลเกี่ยวกับหน่วยความจำสำรอง
psutil.net_io_counters()	แสดงข้อมูลเกี่ยวกับเน็ตเวิร์ค
psutil.users()	แสดงข้อมูลเกี่ยวกับผู้ใช้งาน
psutil.pids()	แสดงข้อมูลเกี่ยวกับโปรเซส

ส่วนที่สองแสดงการคำนวณหน่วยความจำด้วยวิธีทางคณิตศาสตร์

องค์ประกอบของกวิเคราะห์หน่วยความจำที่ใช้โดยวิธีคณิตศาสตร์ มีดังต่อไปนี้

- Instruction Space คือ ขนาดหน่วยความจำที่จำเป็นต้องใช้ขณะคอมไพล์ (Compiler) คอมไพล์โปรแกรมต้นฉบับ
- Data Space คือ ขนาดหน่วยความจำที่จำเป็นต้องใช้สำหรับเก็บข้อมูลค่าคงที่ และตัวแปรที่ใช้ในขณะประมวลผลโปรแกรม ซึ่งแยกออกได้ 2 ประเภท คือ
 - Static memory ขนาดของหน่วยความจำที่ถูกจ่องไว้ ซึ่งจะถูกประมวลผล แน่นอน เช่น หน่วยความจำที่ใช้เก็บค่าคงที่ หรือตัวแปรชนิดอาร์เรย์
 - Dynamic memory ขนาดหน่วยความจำที่ใช้ในการประมวลผลประเภทไม่แน่นอน คือ จะรู้ว่าต้องใช้หน่วยความจำเท่าไรก็ต่อเมื่อโปรแกรมต้องใช้งาน นั้นเอง เช่น การประกาศตัวแปรพอยเตอร์ (Pointer) ในภาษา C หรือการเก็บข้อมูลในรูปแบบลิงค์ลิสต์ที่สามารถเพิ่มหรือลดขนาดการเก็บข้อมูลได้แบบอัตโนมัติโดยไม่ต้องจองพื้นที่หน่วยความจำไว้ก่อนใช้งาน เป็นต้น
- Environment Stack Space คือ หน่วยความจำที่ใช้สำหรับเก็บข้อมูลผลลัพธ์ที่ได้จากการประมวลผล เพื่อรอเวลาที่จะนำกลับไปใช้ใหม่ในโปรแกรม ซึ่งหน่วยความจำประเภทนี้จะเกิดขึ้นเมื่อมีการใช้งานเท่านั้น

ตัวอย่างการวิเคราะห์หน่วยความจำที่ใช้ด้วยวิธีทางคณิตศาสตร์ ดังต่อไปนี้

```
def myFunc(data1, data2):
    temp = 0
    temp = data1 + data2
    return temp
```

จากตัวอย่างโปรแกรมข้างบนได้ประกาศตัวแปรทั้งหมด 3 ตัว คือ data1, data2 และ temp ซึ่งตัวแปรทั้งสามเป็นข้อมูลชนิด integer (ภาษาไทยนั้น integer มีขนาดโดยประมาณคือ 24 ไบต์ โดยใช้

คำสั่ง `sys.getsizeof(i)` ดังนั้นเมื่อคำนวณหาพื้นที่หน่วยความจำที่ใช้ทั้งหมดของฟังก์ชันดังกล่าวเท่ากับ $24 \times 3 = 72$ ไปต่อ ตัวอย่างต่อไปแสดงการคำนวณพื้นที่หน่วยความจำที่ใช้งานในลักษณะการเรียกซ้ำ (Recursion) ดังนี้

```
def Factorial(n):
    if n == 0:
        return 1
    else:
        return n * Factorial(n - 1)
```

จากฟังก์ชัน `Factorial(n)` ด้านบน เมื่อกำหนดให้ `n` มีค่าเท่ากับ 3 จะต้องทำการวนซ้ำเท่ากับ 3 ครั้ง ซึ่งจะใช้พื้นที่หน่วยความจำเท่ากับ $3 \times 24 = 72$ ไปต่อ สรุปได้ว่าฟังก์ชัน `Factorial` จะต้องใช้พื้นที่ของหน่วยความจำเท่ากับ $n \times 24$ ไปต่อต่อไป

4. การประเมินประสิทธิภาพอัลกอริทึมด้วยฟังก์ชันอัตราการเติบโต (Growth-rate functions)

การประเมินประสิทธิภาพอัลกอริทึมด้วยฟังก์ชันอัตราการเติบโตจะมุ่งเน้นการวิเคราะห์เวลาที่ใช้ในการประมวลผลเป็นหลัก หรือเรียกว่า Time complexity analysis โดยเวลาที่อัลกอริทึมใช้ทำงานมี 2 ประเภทคือ เวลาที่ใช้ในการตรวจสอบ (Compile time) และเวลาที่เครื่องคอมพิวเตอร์ใช้ในการประมวลผลอัลกอริทึม (Execution time) ซึ่งขึ้นอยู่กับชนิดข้อมูล จำนวนตัวแปรที่ใช้ และจำนวนลูปเป็นต้น ตัวอย่างที่ 1: แสดงการวิเคราะห์เวลาที่ใช้ฟังก์ชันอัตราการเติบโต ดังต่อไปนี้

ตัวอย่างที่ 1:

<code>n = 0</code>	⇒ กำหนดค่า 1 ครั้ง ①
<code>total = 0</code>	⇒ กำหนดค่า 1 ครั้ง ②
<code>while n <= 30:</code>	⇒ เปรียบเทียบ <code>n + 1</code> ครั้ง ③
<code>total = total + n</code>	⇒ คำนวณ <code>n</code> ครั้ง ④
<code>n = n + 1</code>	⇒ คำนวณ <code>n</code> ครั้ง ⑤
<code>print("Total = ", total)</code>	⇒ แสดงผล 1 ครั้ง ⑥

กำหนดให้ $f(n)$ แทนประสิทธิภาพในการวิเคราะห์เวลาที่ใช้ในการประมวลผล และ n แทนจำนวนรอบในการทำงาน เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้ดังนี้

$$f(n) = ① + ② + ③ + ④ + ⑤ + ⑥$$

$$f(n) = 1 + 1 + (n + 1) + n + n + 1 = 1 + 1 + 1 + 1 + (n + n + n) = 3n + 4$$

$$f(n) = 3n + 4 \quad \dots \dots \dots \textcircled{1}$$

ตัวอย่างที่ 2: เป็นการเขียนสมการฟังก์ชันอัตราการเติบโตชนิดเวียนเกิด ดังนี้

```
def Factorial(n):
    if n == 0:
        return 1
    else: return n * Factorial(n - 1)
```

\Rightarrow ถูกเรียกใช้ n ครั้ง ❶
 \Rightarrow ตรวจสอบว่า n ไม่เท่า 0 ครั้ง ❷
 \Rightarrow คืนค่า 1 ครั้ง ❸
 \Rightarrow เรียกใช้ตัวเอง n ครั้ง ❹

เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้คือ

$$f(n) = ❶ + ❷ + ❸ + ❹ = n + n + 1 + n$$

$$f(n) = 3n + 1 \quad \dots \quad ②$$

ตัวอย่างที่ 3: เป็นการเขียนสมการฟังก์ชันอัตราการเติบโตชนิดวนลูปขั้มลำดับ (งานจะเสร็จเร็วกว่ากำหนด $\frac{n}{2}$) ดังนี้

```
total = 0
for i in range(1, 3, 5, 7, 9):
    total = total + i
    total = total * 2
print("Total = ", total)
```

\Rightarrow กำหนดค่า 1 ครั้ง
 \Rightarrow เพิ่ยบเพิ่ยบ $\frac{n}{2} + 1$ ครั้ง
 \Rightarrow คำนวณ $\frac{n}{2}$ ครั้ง
 \Rightarrow คำนวณ $\frac{n}{2}$ ครั้ง
 \Rightarrow แสดงผล 1 ครั้ง

เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้คือ

$$f(n) = 1 + (\frac{n}{2} + 1) + \frac{n}{2} + \frac{n}{2} + 1 = \frac{3n}{2} + 3 \quad \dots \quad ③$$

ตัวอย่างที่ 4: เป็นการเขียนสมการฟังก์ชันอัตราการเติบโตชนิด Logarithmic ฐานสองดังนี้

สำหรับค่าตัวแปรที่กำหนดที่เป็นเงื่อนไขของลูปจะมีการเพิ่มขึ้นหรือลดลงด้วยการคูณหรือการหารเป็นอัตราเท่าตัวหรือ 2^n

```
total = 0
for i in (2**x for x in range(10)):
    total = total + i
    total = total * 2
print("Total = ", total)
```

\Rightarrow กำหนดค่า 1 ครั้ง
 \Rightarrow เพิ่ยบเพิ่ยบ $\log_2 n + 1$ ครั้ง
 \Rightarrow คำนวณ $\log_2 n$ ครั้ง
 \Rightarrow คำนวณ $\log_2 n$ ครั้ง
 \Rightarrow แสดงผล 1 ครั้ง

ผลลัพธ์ของค่า $i = 1, 2, 4, 8, 16, 32, \dots, 2^n$

เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้คือ

$$f(n) = 1 + (\log_2 n + 1) + \log_2 n + \log_2 n + 1 = 3 \log_2 n + 3 \quad \text{--- (4)}$$

ตัวอย่างที่ 5: เป็นการเขียนสมการฟังก์ชันอัตราการเติบโตชนิดลูปซ้อนดังนี้

<pre>for i in range(10): for i in range(10): total = total + i</pre>	\Rightarrow เปรียบเทียบ $n + 1$ ครั้ง \Rightarrow เปรียบเทียบ $(n + 1) = n^2 + n$ ครั้ง \Rightarrow คำนวณ $n^2 \times n = n^3$ ครั้ง
--	--

เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้คือ

$$f(n) = (n + 1) + (n^2 + n) + n^3 = 2n^3 + 2n + 1 \quad \text{--- (5)}$$

ตัวอย่างที่ 6: เป็นการเขียนสมการฟังก์ชันอัตราการเติบโตชนิด Linear Logarithmic ดังนี้

<pre>for i in range(10): for i in (2**x for x in range(10)): total = total + i</pre>	$\Rightarrow n + 1$ ครั้ง $\Rightarrow n(\log_2 n + 1)$ ครั้ง $\Rightarrow n \log_2 n$ ครั้ง
--	--

เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้คือ

$$f(n) = (n + 1) + n(\log_2 n + 1) + n \log_2 n = n + 2n \log_2 n + 2 \quad \text{--- (5)}$$

ตัวอย่างที่ 7: เป็นการเขียนสมการฟังก์ชันอัตราการเติบโตชนิด Dependent Quadratic ดังนี้

<pre>Total = 0; for (i = 0; i < n; i++) { for (j = 0; j < i; j++) { } }</pre>	$\Rightarrow 1$ ครั้ง $\Rightarrow n + 1$ $\Rightarrow n(\frac{n+1}{2} + 1)$ $\Rightarrow n(\frac{n+1}{2})$ $\Rightarrow n(\frac{n+1}{2})$
---	--

เขียนเป็นสมการฟังก์ชันอัตราการเติบโตได้คือ

Asymptotic notation

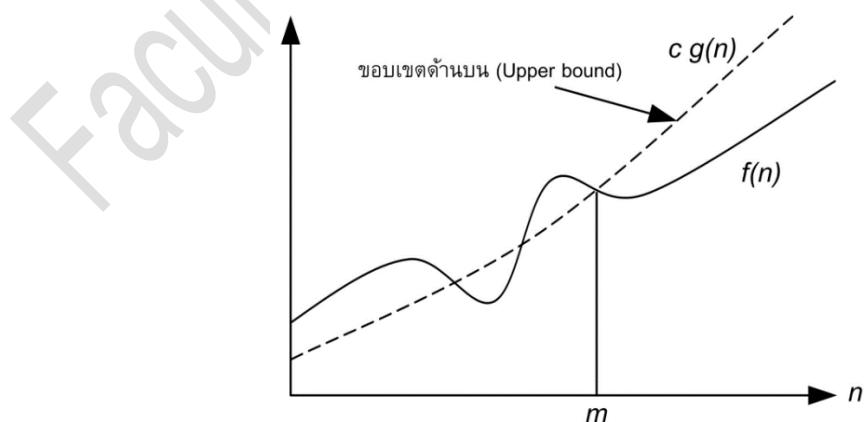
Asymptotic notation คือเครื่องหมายหรือสัญลักษณ์ที่ใช้อธิบายการเจริญเติบโตของฟังชัน หรืออัลกอริทึม (Algorithm Growth Rates) ซึ่งจะใช้ในการวัดประสิทธิภาพของอัลกอริทึม เช่น Big-O, Big-Omega, Big-Teta, Little-o และ Little-omega เป็นต้น แต่ในที่นี้ผู้เขียนจะอธิบายเฉพาะ Big-O เท่านั้น เพราะเป็นวิธีที่นิยมมากที่สุด ซึ่งมีรายละเอียดดังนี้

อัตราการเติบโต Big-O

Big-O เป็นขอบเขตบน (Upper bound) ของประสิทธิภาพที่แย่ที่สุดในการประมวลผลของอัลกอริทึม (Worst case) ซึ่งหมายความว่าอัลกอริทึมนี้จะทำงานไม่แย่ไปกว่า Big-O ของมันแล้ว ซึ่งก็เหมือนกับเป็นตัวบอกถึงประสิทธิภาพของอัลกอริทึมนั้นๆ นั่นเอง หรือเพื่อใช้ในการเปรียบเทียบว่าอัลกอริทึมใดดีกว่ากัน ซึ่ง Big-O มีคุณสมบัติดังนี้คือ

- เป็นการวัดประสิทธิภาพเชิงเวลาที่ใช้ในการประมาณผลของอัลกอริทึม
 - เป็นฟังก์ชันเวลาขอบเขตบนที่ใช้ในการประมาณผล (Asymptotic upper bounds)
 - สัญลักษณ์เป็นตัวโอใหญ่ (O)
 - $O(n)$ หมายถึง พังก์ชันที่จะใช้เวลาในการประมาณผลน้อยกว่าหรือเท่ากับ n ($< n$) เสมอ

นิยาม Big-O: พังก์ชัน $f(n) = O(g(n))$ ก็ต่อเมื่อมีค่าคงที่ m, c ที่ทำให้ $f(n) \leq cg(n)$ เมื่อ $n > m$
ดังรูป 17.1



รูปที่ 17.1 Big-O

จากตัวอย่าง กำหนดให้ $f(n) = 5n^4 - 37n^3 + 13n - 4$ สามารถแสดงการหาค่า Big-O และค่าคงที่ c กับ m ได้ดังนี้

จาก $f(n) = |5n^4 - 37n^3 + 13n - 4| \leq c|n^4|$ โดยที่ $n > m$ (\dots คือค่าจำนวนเต็มบวกเท่านั้น) จะได้ว่า

เมื่อกำหนดให้ $m = 1$ จะทำให้ $n > 1$ สามารถทำให้การคำนวณการถูกต้องได้ดังนี้

$$|5n^4 - 37n^3 + 13n - 4| \leq |5n^4 + 37n^4 + 13n^4 + 4n^4| \quad (\text{เนื่องจาก } 37n^4 > 37n^3)$$

$$\leq 59|n^4|$$

เพราะฉะนั้น $|5n^4 - 37n^3 + 13n - 4| \leq 59|n^4|$ เมื่อ $n > 1$

ดังนั้น $f(n) = 5n^4 - 37n^3 + 13n - 4 \in O(n^4)$

จึงได้ว่า $c = 59$, $n = 1$ และ Big-O คือ $O(n^4)$



Note: สรุปแนวคิดการหา Big-O จะพิจารณาค่าที่มีผลกระทบมากที่สุดเพียงค่าเดียว ซึ่งค่าคงที่ c มีผลน้อยกว่าค่าที่มีผลกระทบมากที่สุด ดังนั้น จึงนำค่าที่มีผลกระทบมากที่สุดเป็นค่าของตัวแปรประสิทธิภาพของ Big-O

สรุปการหาค่า Big-O แบบง่ายๆ

1. ตัดสัมประสิทธิ์ของแต่ละเทอมทิ้ง (ตัดค่าคงที่ทิ้ง)
2. เลือกเทอมที่ใหญ่ที่สุดเก็บไว้เป็นคำตอบ

ตัวอย่างที่ 1: จากตัวอย่างที่ ① $f(n) = 3n + 4$

$$f(n) = n \Rightarrow \text{ตัดสัมประสิทธิ์ของแต่ละเทอมทิ้ง}$$

$$O(f(n)) = n = O(n) \Rightarrow \text{เลือกเทอมใหญ่ที่สุด}$$

ตัวอย่างที่ 2: $f(n) = 3n^4 + 2n^2 + n$

$$f(n) = n^4 + n^2 + n \Rightarrow \text{ตัดสัมประสิทธิ์}$$

$$O(f(n)) = n^4 = O(n^4) \Rightarrow \text{เลือกเทอมใหญ่ที่สุด}$$

ตัวอย่างที่ 3: $f(n) = 10n^3 + n^3 + 10$

$$f(n) = n^3 + n^3 \Rightarrow \text{ตัดสัมประสิทธิ์}$$

$O(f(n)) = n^3 = O(n^3) \Rightarrow$ เลือกเทอมใหญ่ที่สุด

ตัวอย่างที่ 4: $f(n) = 100$

$$O(f(n)) = 1 = O(1)$$

ตัวอย่างที่ 5: $f(n) = 100n + 1$

$$f(n) = n$$

$$O(f(n)) = n = O(n)$$

ตัวอย่างที่ 6: $f(n) = 20n\log_n + 5n$

$$f(n) = n\log_n + n \Rightarrow \text{ตัดสัมประสิทธิ์}$$

$$O(f(n)) = O(n\log_n) \Rightarrow$$
 เลือกเทอมใหญ่ที่สุด

ตัวอย่างที่ 7: กำหนดให้แต่ละโปรแกรมทำงานดังนี้คือ โปรแกรม 1 $\Rightarrow f(n) = 3n^2 + 2n$,

โปรแกรม 2 $\Rightarrow f(n) = 2\log_2 n + 6n + n$, โปรแกรมที่ 3 $\Rightarrow f(n) = n + n\log_2 n + 4n + 9$ จงหาค่า Big-O และประเมินว่าโปรแกรมใดมีประสิทธิภาพการทำงานจากดีที่สุดไปยังแย่ที่สุด

โปรแกรมที่ 1 $\Rightarrow f(n) = 3n^2 + 2n = n^2 + n$

$$= O(n^2)$$

โปรแกรมที่ 2 $\Rightarrow f(n) = 2\log_2 n + 6n + n = \log_2 n + n + n$

$$= O(n)$$

โปรแกรมที่ 3 $\Rightarrow f(n) = n + n\log_2 n + 4n + 9 = n + n\log_2 n + n$

$$= O(n\log_2 n)$$

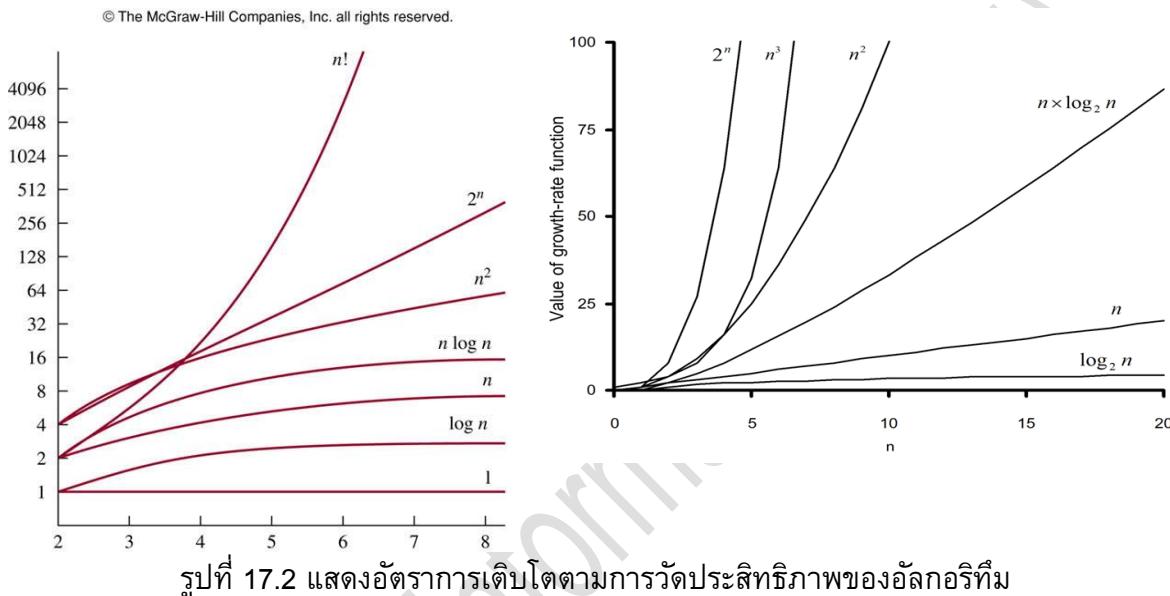
เพราะจะนั้น $O(n)$ (โปรแกรมที่ 2 ดีที่สุด) $< O(n\log_2 n)$ (โปรแกรมที่ 3) $< O(n^2)$ (โปรแกรมที่ 1 แย่ที่สุด)

ตารางที่ 17.1 แสดงบทสรุปของอัตราการเติบโตตามการวัดประสิทธิภาพของอัลกอริทึม

$f(n)$	แบบการหับตัวดำเนินการ	ประสิทธิภาพ
c	ค่าคงที่ (Constant)	เร็วที่สุด
$\log_2 n$	พังชันลอการิทึม (Logarithm loops)	ค่อนข้างเร็ว
n	พังชันเชิงเส้น (Linear loops)	

$n \log_2 n$	ฟังชันลอการิทึมเชิงเส้น (Linear Logarithm)	
n^2	ฟังชันกำลังสอง (Quadratic)	ปานกลาง
n^3	ฟังชันกำลังสาม (Cubic)	
n^k	ฟังชันโพลีโนเมียล (Polynomial)	
2^n	ฟังชันเอ็กโพเนนเชียล (Exponential)	ค่อนข้างช้า
$n!$	ฟังชันแฟกทอเรียล (Factorial)	ช้าที่สุด

จากตารางที่ 7.1 สามารถแสดงเป็นกราฟเชิงเส้นได้ดังรูปที่ 17.2



รูปที่ 17.2 แสดงอัตราการเติบโตตามการวัดประสิทธิภาพของอัลกอริทึม

5. การวิเคราะห์ Best-case, Worst-case และ Average-case

Base-case: การวิเคราะห์หาประสิทธิภาพที่ดีที่สุดในการประมวลผลของอัลกอริทึม เช่น การค้นหาข้อมูลในอาร์เรย์ และเจอข้อมูลทันทีเมื่อทำการตรวจสอบในครั้งแรก

Worst case: การวิเคราะห์หาประสิทธิภาพที่แย่ที่สุดในการประมวลผลของอัลกอริทึม เช่น การค้นหาข้อมูลในอาร์เรย์ และเจอข้อมูลในการตรวจสอบครั้งสุดท้าย เป็นต้น แสดงว่ากรณีนี้เป็นกรณีที่แย่ที่สุด เพราะต้องตรวจสอบข้อมูลจนถึงครั้งสุดท้ายจึงจะพบข้อมูลที่ต้องการ

Average-case: การหาค่าเฉลี่ยของเวลาที่ใช้ประมวลผลของอัลกอริทึม โดยนำเวลาที่ดีที่สุด + เวลาที่แย่ที่สุด และหารด้วย 2

6. การจัดเรียงข้อมูล (Sorting)

Sorting หมายถึง การจัดเรียงข้อมูลให้มีการเรียงลำดับตามที่ผู้ใช้งานต้องการ โดยจะทำการเรียงข้อมูลจากค่าที่น้อยไปมาก หรือเรียงข้อมูลจากมากไปน้อยก็ได้ เช่น การเรียงลำดับตามความสูง จัด

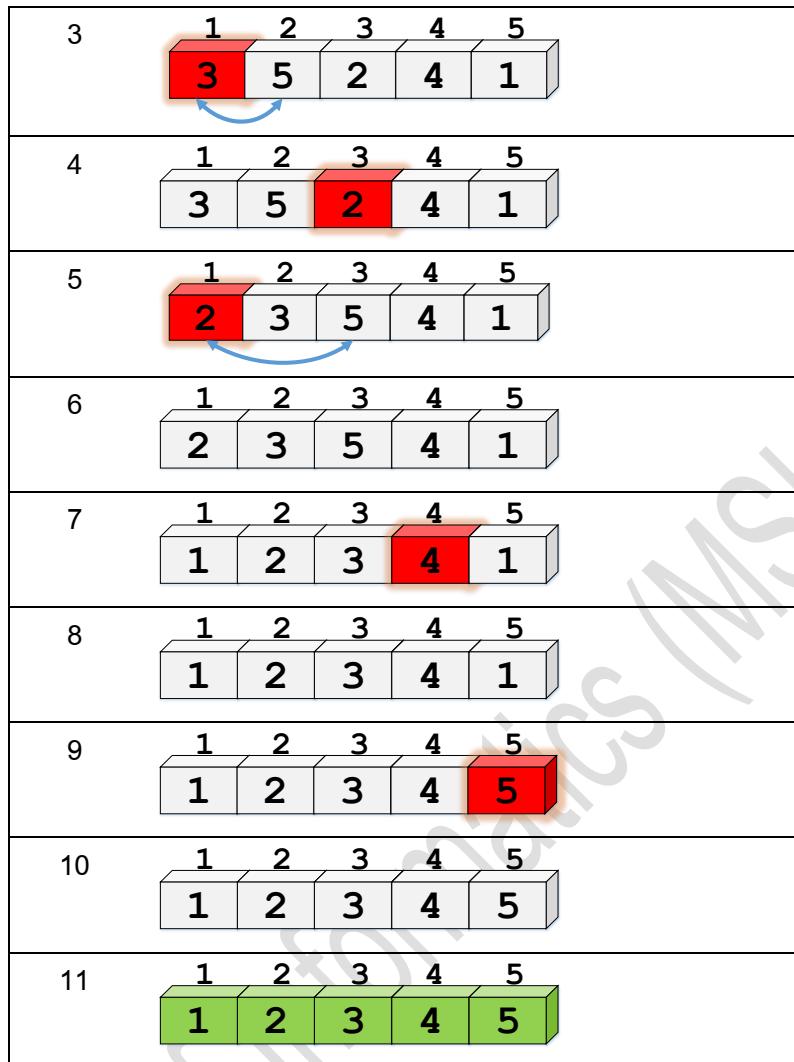
เรียงลำดับตามรหัสนักศึกษา การเรียงรายนามผู้ใช้โทรศัพท์ตามลำดับข้อมูลตัวอักษร การจัดเรียงตัวอักษรในพจนานุกรม การเรียงลำดับรายชื่อที่มีคะแนนสูงสุดไปคำสุด เป็นต้น สาเหตุที่ทำให้ต้องมีการจัดเรียงข้อมูลก็เพราะว่าง่ายต่อการค้นหา ถ้าหากมีข้อมูลจำนวนมากๆ แล้วไม่ทำการเรียงข้อมูล ก็จะทำให้เสียเวลาในการค้นหาข้อมูลเป็นอย่างมาก วิธีการจัดเรียงข้อมูลในระบบคอมพิวเตอร์แบ่งเป็น 2 ประเภทใหญ่ๆ คือ

- การเรียงข้อมูลภายใน (Internal sorting) จะใช้กับข้อมูลที่มีขนาดไม่เกินพื้นที่หน่วยความจำที่มีอยู่ในระบบ โดยเรียงข้อมูลภายในหน่วยความจำของเครื่องคอมพิวเตอร์ โดยไม่ต้องใช้หน่วยความจำสำรอง เช่น ดิสก์ เทป เป็นต้น ซึ่งมีวิธีการจัดเรียงลำดับข้อมูลภายในมีหลายวิธี ได้แก่ Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Shell Sort, Heap Sort และ Radix Sort เป็นต้น
- การเรียงข้อมูลภายนอก (External sorting) จะใช้กับข้อมูลที่มีขนาดใหญ่เกินกว่าที่จะเก็บลงในหน่วยความจำได้หมดภายในครั้งเดียว และจะใช้หน่วยความจำจากภายนอกแทน เช่น ดิสก์ เทป สำหรับเก็บข้อมูลชั่วคราวที่ได้รับการเรียงข้อมูลแล้ว ซึ่งมีวิธีการจัดเรียงลำดับข้อมูลภายนอกหลายวิธี ได้แก่ Merge Sort, Run List, การเรียงข้อมูลบนดิสก์ และการเรียงข้อมูลบนเทป เป็นต้น

1. การจัดเรียงข้อมูลแบบ Insertion Sort

หลักการ: มีลักษณะคล้ายกับการจัดไฟในเมืองของผู้เล่น คือ เมื่อผู้เล่นได้ไฟใบใหม่เพิ่มขึ้นมา จะนำไฟใบนั้นไปแทรกในตำแหน่งที่เหมาะสม ทำให้ไฟในเมืองบางส่วนต้องย้ายตำแหน่งออกไป ซึ่งการจัดเรียงลำดับข้อมูลแบบแทรกนี้ จะเริ่มพิจารณาข้อมูลในตำแหน่งที่ 2 เป็นต้นไป เช่น ผู้เล่นมีไฟหมายเลข 3, 5 และ 9 อยู่ในเมือง เมื่อได้ไฟใบใหม่มาเป็นเลข 4 ผู้เล่นจะแทรกไฟไปดังกล่าวระหว่างไฟหมายเลข 3 และ 5 เป็นต้น จากนั้นล่างแสดงการจัดเรียงข้อมูลจากน้อยไปมากโดยใช้อัลกอริทึมแบบ Insertion sort

ลำดับที่	แสดงการจัดเรียงลำดับ										
1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr> <tr> <td>5</td><td>3</td><td>2</td><td>4</td><td>1</td></tr> </table>	1	2	3	4	5	5	3	2	4	1
1	2	3	4	5							
5	3	2	4	1							
2	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr> <tr> <td>5</td><td>3</td><td style="background-color: red;">2</td><td>4</td><td>1</td></tr> </table>	1	2	3	4	5	5	3	2	4	1
1	2	3	4	5							
5	3	2	4	1							



จากรูปด้านบนแสดงการทำงานของ Insertion Sort ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

Program Example 17.5: Insertion Sort

```

1 def insertion(data):
2     for i in range(1, len(data)):
3         temp = data[i]
4         j = i
5         while (temp < data[j-1] and j>0):
6             data[j] = data[j-1]
7             j -= 1
8         data[j] = temp
9
10 data = [6, 1, 7, 9, 2, 8, 5, 4, 3]
11 print("The data before sorting = ", data)
12 insertion(data)
13 print("The data after sorting = ", data)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



```

The data before sorting = [6, 1, 7, 9, 2, 8, 5, 4, 3]
The data after sorting = [1, 2, 3, 4, 5, 6, 7, 8, 9]

```

การจัดเรียงลำดับโดยการแทรก จะมีการจัดเรียงลำดับข้อมูลทั้งหมด $n - 1$ รอบ โดยแต่ละรอบนั้น จำนวนการเปรียบเทียบจะไม่แน่นอน เพราะในแต่ละรอบการเปรียบเทียบจะสิ้นสุดเมื่อไม่มีการสลับตำแหน่งของข้อมูล เมื่อพิจารณาจำนวนการเปรียบเทียบข้อมูลแบ่งเป็น 3 กรณีคือ

1. กรณีเดี๋ยสุด (Base-case) ข้อมูลถูกจัดเรียงลำดับเรียบร้อยแล้ว การเปรียบเทียบในกรณีนี้แต่ละรอบจะมีการเปรียบเทียบข้อมูลเพียงครั้งเดียวเท่านั้น เพราะฉะนั้นจำนวนการเปรียบเทียบข้อมูลจะเท่ากับ $n - 1$ ครั้ง ดังนั้น BigO = O(n)
2. กรณีแย่ที่สุด (Worst-case) ข้อมูลถูกจัดเรียงลำดับในตำแหน่งที่สลับกัน เช่น เรียงลำดับค่าข้อมูลจากมากไปหาน้อย (ในกรณีที่ต้องการจัดเรียงลำดับจากน้อยไปมาก) ในกรณีนี้แต่ละรอบจะมีจำนวนการเปรียบเทียบข้อมูลดังนี้

รอบที่ 1 เปรียบเทียบทั้งหมดจำนวน 1 ครั้ง

รอบที่ 2 เปรียบเทียบทั้งหมดจำนวน 2 ครั้ง

รอบที่ 3 เปรียบเทียบทั้งหมดจำนวน 3 ครั้ง

.....

รอบที่ $n - 2$ เปรียบเทียบทั้งหมดจำนวน $n - 2$ ครั้ง

รอบที่ $n - 1$ เปรียบเทียบทั้งหมดจำนวน $n - 1$ ครั้ง

ดังนั้น จำนวนการเปรียบเทียบทั้งหมดเท่ากับ

$$1 + 2 + 3 + \dots + (n - 2) + (n - 1) = \frac{n(n - 1)}{2} = \frac{n^2 - n}{2}$$

ฉะนั้น BigO ในกรณี West-case ของ Insertion Sort เท่ากับ $O(n^2)$

3. กรณีเฉลี่ย (Average-case) จำนวนการเปรียบเทียบข้อมูลทั้งหมด สามารถคำนวณได้จากนำเอาจำนวนการเปรียบเทียบในกรณีที่เดี๋ยสุดและกรณีที่แย่ที่สุดมารวมกันและหารด้วย 2 ดังนี้

$$\frac{(n - 1) + \frac{n^2 - n}{2}}{2} = \frac{\frac{(2n - 2) + n^2 - n}{2}}{2} = \frac{\frac{n^2 + n - 2}{2}}{2} = \frac{n^2 + n - 2}{2} \times \frac{1}{2} = \frac{n^2 + n - 2}{4}$$

ฉะนั้น BigO ในกรณี Average-case ของ Insertion Sort เท่ากับ $O(n^2)$

2. การจัดเรียงข้อมูลแบบ Selection Sort

หลักการ: จดจำตำแหน่งข้อมูลที่มีค่าน้อยที่สุด (กรณีเรียงข้อมูลจากน้อยไปมาก) ในแต่ละรอบและนำข้อมูลในตำแหน่งดังกล่าวมาแลกกับข้อมูลในตำแหน่งที่ 1, 2, 3, ..., n - 1 ตามลำดับ

จากรูปด้านล่างแสดงการจัดเรียงข้อมูลจากน้อยไปมากโดยใช้อัลกอริทึมแบบ Selection Sort

ลำดับที่	แสดงการจัดเรียงลำดับ
1	1 2 3 4 5 5 1 2 4 3
2	1 2 3 4 5 5 1 2 4 3
3	1 2 3 4 5 1 5 2 4 3
4	1 2 3 4 5 1 5 2 4 3
5	1 2 3 4 5 1 5 2 4 3
6	1 2 3 4 5 1 2 5 4 3
7	1 2 3 4 5 1 2 5 4 3
8	1 2 3 4 5 1 2 5 4 3
9	1 2 3 4 5 1 2 3 4 5
10	1 2 3 4 5 1 2 3 4 5
11	1 2 3 4 5 1 2 3 4 5
12	1 2 3 4 5 1 2 3 4 5
13	1 2 3 4 5 1 2 3 4 5
14	1 2 3 4 5 1 2 3 4 5

จากรูปด้านบนแสดงการทำงานของ Selection Sort ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

Program Example 17.6: Selection Sort

```

1 def selection(data):
2     for i in range(0, len(data)-1):
3         indexOfMin = i
4         for j in range(i+1, len(data)):
5             if (data[j] < data[indexOfMin]):
6                 indexOfMin = j
7             temp = data[i]
8             data[i] = data[indexOfMin]
9             data[indexOfMin] = temp
10
11 data = [6, 1, 7, 9, 2, 8, 5, 4, 3]
12 print("The data before sorting = ", data)
13 selection(data)
14 print("The data after sorting = ", data)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



The data before sorting = [6, 1, 7, 9, 2, 8, 5, 4, 3]
 The data after sorting = [1, 2, 3, 4, 5, 6, 7, 8, 9]

ในการนี้ที่มีข้อมูล N ตัว การเรียงลำดับข้อมูลแบบ Selection Sort จะมีการค้นหาทั้งหมด N - 1 ครั้ง และ มีการสลับที่กันจริงไม่เกิน N - 1 ครั้ง โดยในแต่ละรอบจะมีการเปรียบเทียบข้อมูลดังนี้

- กรณีเดี๋ยสุด (Base-case) มีค่า BigO = O(n^2)
- กรณีแย่ที่สุด (Worst-case) ข้อมูลถูกจัดเรียงลำดับในตำแหน่งที่สลับกัน เช่น เรียงลำดับค่า ข้อมูลจากมากไปหาน้อย (ในการนี้ที่ต้องการจัดเรียงลำดับจากน้อยไปมาก) ในกรณีนี้แต่ละ รอบจะมีจำนวนการเปรียบเทียบข้อมูลดังนี้

รอบที่ 1 เปรียบเทียบทั้งหมดจำนวน n - 1 ครั้ง

รอบที่ 2 เปรียบเทียบทั้งหมดจำนวน n - 2 ครั้ง

รอบที่ 3 เปรียบเทียบทั้งหมดจำนวน n - 3 ครั้ง

.....

รอบที่ n - 2 เปรียบเทียบทั้งหมดจำนวน 2 ครั้ง

รอบที่ n - 1 เปรียบเทียบทั้งหมดจำนวน 1 ครั้ง

ดังนั้น จำนวนการเปรียบเทียบทั้งหมดเท่ากับ

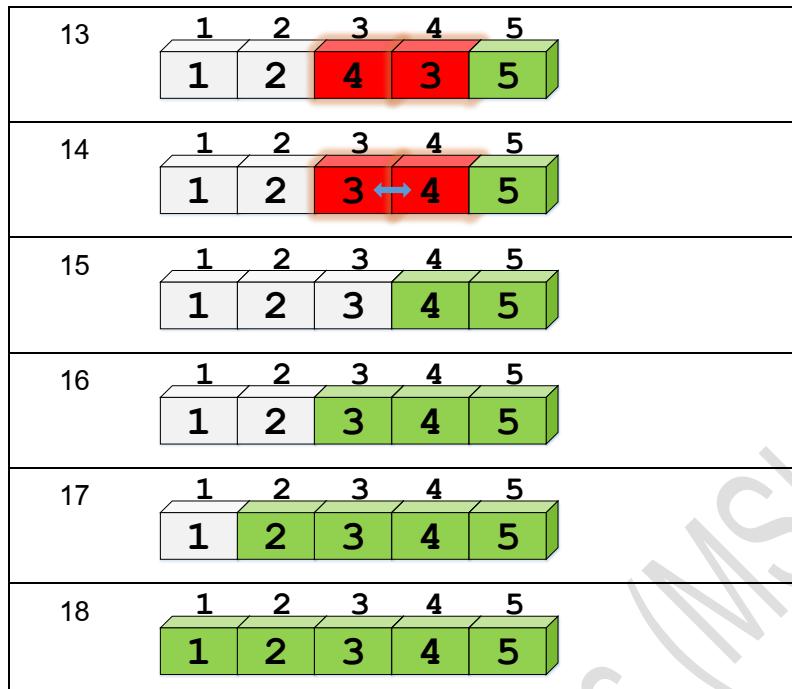
$$1 + 2 + 3 + \dots + (n - 2) + (n - 1) = \frac{n(n - 1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

- กรณีเฉลี่ย (Average-case) มีค่า BigO = O(n^2)

3. การจัดเรียงข้อมูลแบบ Bubble Sort

หลักการ: เปรียบเทียบข้อมูลในตำแหน่งที่อยู่ติดกันทีละคู่ ถ้าข้อมูลที่เปรียบเทียบไม่อยู่ในตำแหน่งที่ต้องการแล้วให้ทำการสลับที่กันระหว่างข้อมูล 2 ตัวนั้น ทำเช่นนี้จนกระทั่งเปรียบเทียบครบทุกตัว ซึ่งคือ $N - 1$ ครั้ง จากรูปด้านล่างแสดงการจัดเรียงข้อมูลจากน้อยไปมากโดยใช้อัลกอริทึมแบบ Bubble Sort

ลำดับที่	แสดงการจัดเรียงลำดับ										
1	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">5</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	5	1	2	4	3
1	2	3	4	5							
5	1	2	4	3							
2	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">5</td><td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	5	1	2	4	3
1	2	3	4	5							
5	1	2	4	3							
3	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">5</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	1	5	2	4	3
1	2	3	4	5							
1	5	2	4	3							
4	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="background-color: red; color: white; text-align: center;">5</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	1	5	2	4	3
1	2	3	4	5							
1	5	2	4	3							
5	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="background-color: red; color: white; text-align: center;">2</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	1	2	5	4	3
1	2	3	4	5							
1	2	5	4	3							
6	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="background-color: red; color: white; text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	1	2	5	4	3
1	2	3	4	5							
1	2	5	4	3							
7	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	1	2	4	5	3
1	2	3	4	5							
1	2	4	5	3							
8	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="background-color: red; color: white; text-align: center;">5</td><td style="text-align: center;">3</td></tr> </table>	1	2	3	4	5	1	2	4	5	3
1	2	3	4	5							
1	2	4	5	3							
9	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">5</td></tr> </table>	1	2	3	4	5	1	2	4	3	5
1	2	3	4	5							
1	2	4	3	5							
10	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="background-color: green; color: white; text-align: center;">5</td></tr> </table>	1	2	3	4	5	1	2	4	3	5
1	2	3	4	5							
1	2	4	3	5							
11	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="background-color: red; color: white; text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="background-color: green; color: white; text-align: center;">5</td></tr> </table>	1	2	3	4	5	1	2	4	3	5
1	2	3	4	5							
1	2	4	3	5							
12	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">2</td><td style="text-align: center;">3</td><td style="text-align: center;">4</td><td style="text-align: center;">5</td></tr> <tr> <td style="text-align: center;">1</td><td style="background-color: red; color: white; text-align: center;">2</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="background-color: green; color: white; text-align: center;">5</td></tr> </table>	1	2	3	4	5	1	2	4	3	5
1	2	3	4	5							
1	2	4	3	5							



จากรูปด้านบนแสดงการทำงานของ Bubble Sort ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

Program Example 17.7: Bubble Sort

```

1 def bubble(data):
2     for i in range(0, len(data)-1):
3         for j in range(len(data)-1, i, -1):
4             if (data[j] < data[j-1]):
5                 temp = data[j]
6                 data[j] = data[j-1]
7                 data[j-1] = temp
8
9     data = [6, 1, 7, 9, 2, 8, 5, 4, 3]
10    print("The data before sorting = ", data)
11    bubble(data)
12    print("The data after sorting = ", data)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



```

The data before sorting = [6, 1, 7, 9, 2, 8, 5, 4, 3]
The data after sorting = [1, 2, 3, 4, 5, 6, 7, 8, 9]

```

จากโปรแกรมสังเกตว่าข้อมูลมีการสลับที่กันไปเรื่อยๆ จนได้ข้อมูลที่มีการเรียงลำดับเป็นที่เรียบร้อยแล้ว ซึ่งมีการจัดเรียง $n - 1$ รอบ โดยในแต่ละรอบจะมีการเปรียบเทียบข้อมูลเป็น $n - 1, n - 2, n - 3, \dots, 3, 2, 1$ ดังนั้น จำนวนการเปรียบเทียบทั้งหมดคือ

1. กรณีดีที่สุด (Base-case) มีค่า BigO = O(n)
2. กรณีแย่ที่สุด (Worst-case) มีค่า BigO เท่ากับ

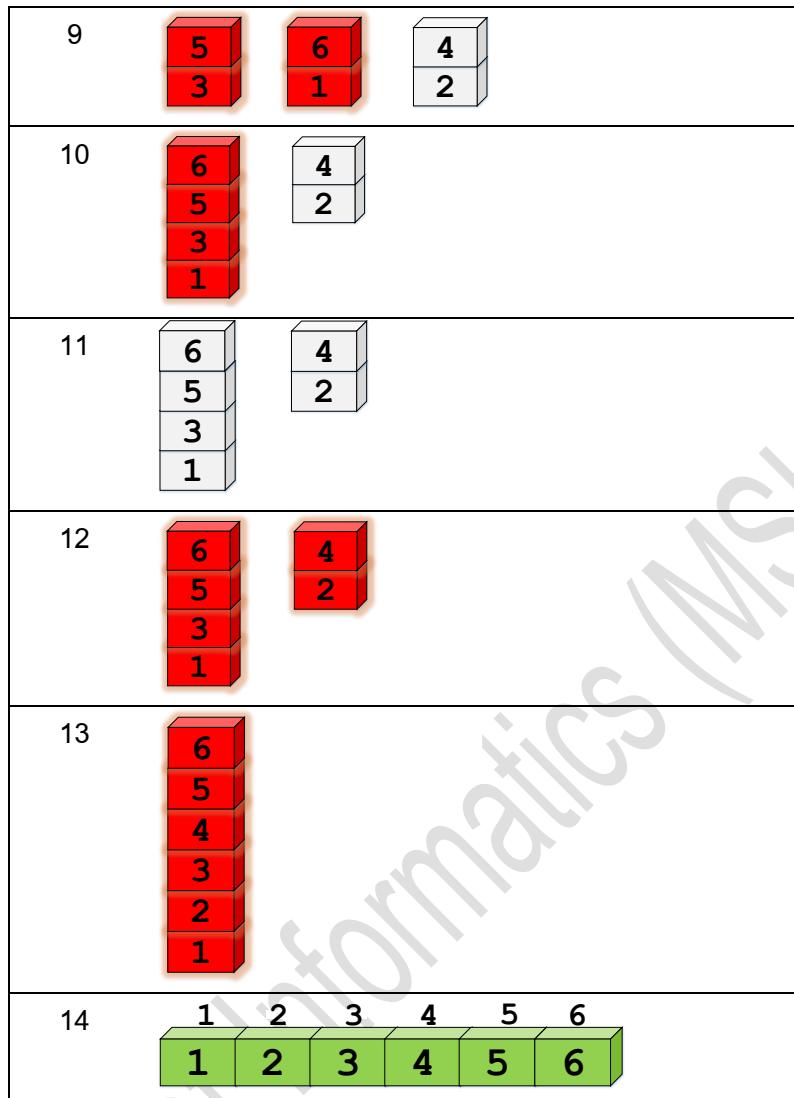
$$1 + 2 + 3 + \dots + (n - 2) + (n - 1) = \frac{n(n - 1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

3. กรณีเฉลี่ย (Average-case) มีค่า BigO = O(n²)

4. การจัดเรียงข้อมูลแบบ Merge Sort

หลักการ: เป็นวิธีการเรียงข้อมูลที่มีความสำคัญมาก เพราะเป็นวิธีสำคัญในการเรียงข้อมูลขนาดใหญ่ (การเรียงแบบภายนอก) วิธีการเรียงจะเริ่มกระทำครั้งละ 2 ค่า ซึ่งจะได้ลิสต์ย่อยจำนวน $n/2$ ลิสต์ แต่ละลิสต์มี 2 ค่า จากนั้นจะทำการ merge ต่ออีกรอบละ 2 ลิสต์แล้วจะได้ลิสต์ที่เรียงแล้ว จำนวน $n/4$ ลิสต์ แต่ละลิสต์มี 4 ค่า ทำเช่นนี้ไปเรื่อยๆ จนในที่สุดจะทำการ merge 2 ลิสต์สุดท้ายเข้าด้วยกัน จะได้ลิสต์ที่เรียงเรียบร้อยแล้ว จากรูปด้านล่างแสดงการจัดเรียงข้อมูลจากน้อยไปมากโดยใช้อัลกอริทึมแบบ Bubble Sort

ลำดับที่	แสดงการจัดเรียงลำดับ												
1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td> </tr> <tr> <td>5</td><td>3</td><td>6</td><td>1</td><td>4</td><td>2</td> </tr> </table>	1	2	3	4	5	6	5	3	6	1	4	2
1	2	3	4	5	6								
5	3	6	1	4	2								
2	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td> </tr> <tr> <td>5</td><td>3</td><td>6</td><td>1</td><td>4</td><td>2</td> </tr> </table>	1	2	3	4	5	6	5	3	6	1	4	2
1	2	3	4	5	6								
5	3	6	1	4	2								
3	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>3</td><td>4</td><td>5</td><td>6</td> </tr> <tr> <td>6</td><td>1</td><td>4</td><td>2</td> </tr> </table> 	3	4	5	6	6	1	4	2				
3	4	5	6										
6	1	4	2										
4	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>3</td><td>4</td><td>5</td><td>6</td> </tr> <tr> <td>6</td><td>1</td><td>4</td><td>2</td> </tr> </table> 	3	4	5	6	6	1	4	2				
3	4	5	6										
6	1	4	2										
5	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>5</td><td>6</td> </tr> <tr> <td>4</td><td>2</td> </tr> </table> 	5	6	4	2								
5	6												
4	2												
6	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>5</td><td>6</td> </tr> <tr> <td>4</td><td>2</td> </tr> </table> 	5	6	4	2								
5	6												
4	2												
7	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>5</td><td>6</td> </tr> <tr> <td>3</td><td>1</td> </tr> <tr> <td>4</td><td>2</td> </tr> </table>	5	6	3	1	4	2						
5	6												
3	1												
4	2												
8	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>5</td><td>6</td><td>4</td> </tr> <tr> <td>3</td><td>1</td><td>2</td> </tr> </table>	5	6	4	3	1	2						
5	6	4											
3	1	2											



จากรูปด้านบนแสดงการทำงานของ Merge Sort ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

Program Example 17.8: Merge Sort

```

1  def mergeSort(data):
2      if len(data) < 2: return data
3      m = len(data) // 2
4      return merge(mergeSort(data[:m]), mergeSort(data[m:]))
5
6  def merge(l, r):
7      result = []
8      i = j = 0
9      while i < len(l) and j < len(r):
10         if l[i] < r[j]:
11             result.append(l[i])
12             i += 1
13         else:
14             result.append(r[j])
15             j += 1
16     result.extend(l[i:])
17     result.extend(r[j:])
18     return result

```

```

19
20 data = [6, 1, 7, 9, 2, 8, 5, 4, 3]
21 print("The data before sorting = ", data)
22 data = mergeSort(data)
23 print("The data after sorting = ", data)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

The data before sorting = [6, 1, 7, 9, 2, 8, 5, 4, 3]
The data after sorting = [1, 2, 3, 4, 5, 6, 7, 8, 9]

```

จากโปรแกรมจำนวนการเปรียบเทียบทั้งหมดคือ

1. กรณีดีที่สุด (Base-case) มีค่า BigO = O(nlog₁₀(n))
2. กรณีแย่ที่สุด (Worst-case) มีค่า BigO = O(nlog₁₀(n))
3. กรณีเฉลี่ย (Average-case) มีค่า BigO = O(nlog₁₀(n))

7. การค้นหาข้อมูล (Searching)

การค้นหาข้อมูลมีอยู่หลายวิธีด้วยกัน แต่ที่นิยมใช้และควรที่จะต้องทำความเข้าใจได้แก่

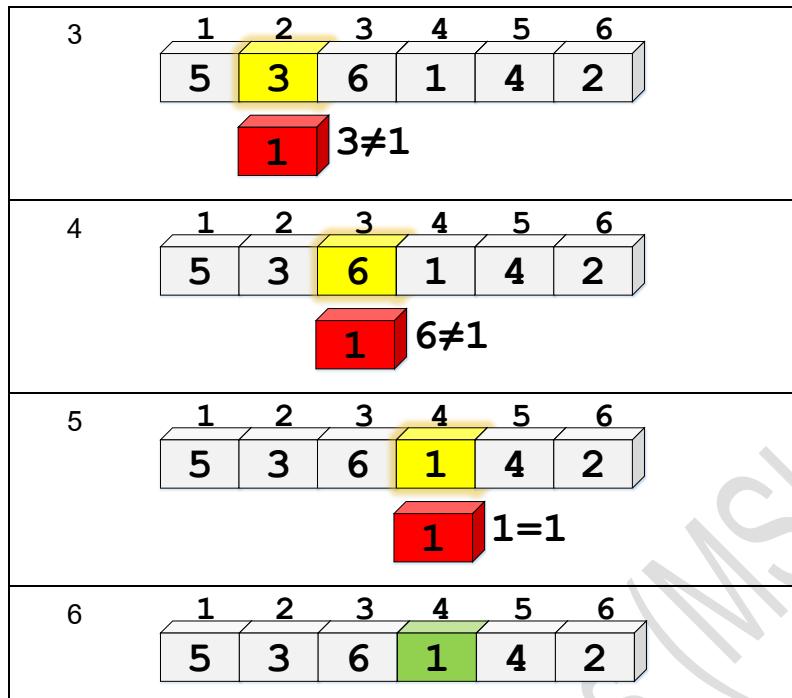
- การค้นหาข้อมูลแบบลำดับ (Sequential Search)
- การค้นหาข้อมูลแบบพื้นครึ่ง (Binary Search)

1. การค้นหาข้อมูลแบบลำดับ (Sequential Search)

หลักการ: การค้นหาข้อมูลแบบลำดับหรือเรียกอีกชื่อหนึ่งว่า Linear Search เป็นการค้นหาข้อมูลที่มีลักษณะการทำงานแบบเรียงตามลำดับ เป็นวิธีที่ง่ายที่สุด ในการค้นหาข้อมูลแบบนี้จะทำการตรวจสอบข้อมูลที่ต้องการโดยไปทีล่ะ 1 ข้อมูลตามลำดับ ทำอย่างนี้ไปเรื่อยๆ จนกว่าจะพบข้อมูล (Key) ตามที่ต้องการหรือหมดข้อมูลที่ต้องการค้นหา จากรูปด้านล่างแสดงการค้นหาข้อมูลโดยใช้อัลกอริทึมแบบ Sequential Search

ลำดับที่	แสดงการจัดเรียงลำดับ					
1	1	2	3	4	5	6
	5	3	6	1	4	2

ลำดับที่	แสดงการจัดเรียงลำดับ					
2	1	2	3	4	5	6
	5	3	6	1	4	2
	1	5≠1				



จากรูปด้านบนแสดงการทำงานของ Sequential Search ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

Program Example 17.9: Sequential Search

```

1 def sequentialSearch(data, key):
2     for i in range(len(data)):
3         if key == data[i]:
4             return i
5     return -1
6
7 data = [6, 1, 7, 9, 2, 8, 5, 4, 3]
8 print("Data source = ", data)
9 print("Would like to search the 8 number.")
10 position = sequentialSearch(data, 8)
11 print("The position of the 8 number is ", position)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT Data source = [6, 1, 7, 9, 2, 8, 5, 4, 3]
Would like to search the 8 number.
The position of the 8 number is 5

การทำงานของค้นหาข้อมูลแบบนี้ จะทำการรับค่าข้อมูล (Key) สำหรับค้นหาแล้วนำไปค้นหาในข้อมูลที่เก็บอยู่ สำหรับวิธีการนี้สามารถสรุปประสิทธิภาพในการค้นหาได้เป็น 3 กรณี คือ

- กรณีที่ค้นหามาตรฐาน โปรแกรมทำการเปรียบเทียบข้อมูลเพียง 1 ครั้งเท่านั้น ถ้าข้อมูลที่ต้องการค้นหาอยู่อันดับแรกของข้อมูลทั้งหมด จะนับค่า BigO = O(1)
- กรณีที่ไม่พบ โปรแกรมทำการเปรียบเทียบข้อมูลจำนวน n ครั้ง ($n = \text{จำนวนข้อมูลทั้งหมด}$) ถ้าข้อมูลที่ต้องการค้นหาอยู่อันดับสุดท้ายของข้อมูลทั้งหมด จะนับค่า BigO = O(n)
- กรณีเฉลี่ยทั่วไป โปรแกรมทำการเปรียบเทียบข้อมูลประมาณ $n/2$ ครั้ง

2. การค้นหาข้อมูลแบบพับครึ่ง (Binary Search)

หลักการ: การค้นหาข้อมูลแบบพับครึ่งนี้ได้ถูกคิดคันขึ้นมาเพื่อแก้ไขข้อเสียของการค้นหาข้อมูลแบบลำดับในกรณีที่ข้อมูลที่ต้องการค้นหาอยู่ตัวท้ายๆ ของข้อมูล แต่ข้อกำหนดของการค้นหาข้อมูลแบบพับครึ่งนี้จะสามารถทำงานได้กับข้อมูลที่มีการจัดเรียงเรียบร้อยแล้ว (เรียงจากมากไปน้อย หรือเรียงจากน้อยไปมากก็ได้)

หลักการค้นหาข้อมูลแบบพับครึ่งจะทำงานโดยการแบ่งข้อมูลออกเป็น 2 ส่วน แล้วนำค่ากลางมาเปรียบเทียบกับข้อมูล (Key) ที่ต้องการค้นหา ถ้าข้อมูลที่ทำการค้นหาระบบเรียงลำดับจากน้อยไปมากก็ เมื่อเปรียบเทียบแล้วข้อมูลที่ต้องการค้นหามีค่ามากกว่า แสดงว่าต้องไปค้นหาข้อมูลต่อในส่วนข้อมูลครึ่งหลังต่อ จากนั้นให้นำข้อมูลครึ่งหลังพับครึ่งหาค่ากลางอีก ทำอย่างนี้ไปเรื่อยๆ จนกว่าจะได้ข้อมูลที่ต้องการ ดังตัวอย่างต่อไปนี้

ลำดับที่		แสดงการจัดเรียงลำดับ																	
1		<table border="1"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td> </tr> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td> </tr> </table>						1	2	3	4	5	6	1	2	3	4	5	6
1	2	3	4	5	6														
1	2	3	4	5	6														
2		<table border="1"> <tr> <td>1</td><td>2</td><td>3</td> <td>4</td><td>5</td><td>6</td> </tr> <tr> <td>1</td><td>2</td><td style="background-color: yellow;">3</td> <td>4</td><td>5</td><td>6</td> </tr> </table> $ (1+6)/2 $						1	2	3	4	5	6	1	2	3	4	5	6
1	2	3	4	5	6														
1	2	3	4	5	6														
3		<table border="1"> <tr> <td>1</td><td>2</td><td style="background-color: yellow;">3</td> <td>4</td><td>5</td><td>6</td> </tr> <tr> <td>1</td><td>2</td><td style="background-color: yellow;">3</td> <td>4</td><td>5</td><td>6</td> </tr> </table> 5 3 < 5						1	2	3	4	5	6	1	2	3	4	5	6
1	2	3	4	5	6														
1	2	3	4	5	6														
4		<table border="1"> <tr> <td>1</td><td>2</td><td>3</td> <td>4</td><td>5</td><td>6</td> </tr> <tr> <td>1</td><td>2</td><td style="background-color: yellow;">3</td> <td>4</td><td style="background-color: yellow;">5</td> <td>6</td> </tr> </table> $ (4+6)/2 $						1	2	3	4	5	6	1	2	3	4	5	6
1	2	3	4	5	6														
1	2	3	4	5	6														
5		<table border="1"> <tr> <td>1</td><td>2</td><td>3</td> <td>4</td><td>5</td><td>6</td> </tr> <tr> <td>1</td><td>2</td><td style="background-color: yellow;">3</td> <td>4</td><td style="background-color: yellow;">5</td> <td>6</td> </tr> </table> 5 5 = 5						1	2	3	4	5	6	1	2	3	4	5	6
1	2	3	4	5	6														
1	2	3	4	5	6														
6		<table border="1"> <tr> <td>1</td><td>2</td><td>3</td> <td>4</td><td>5</td><td>6</td> </tr> <tr> <td>1</td><td>2</td><td style="background-color: yellow;">3</td> <td>4</td><td style="background-color: green;">5</td> <td>6</td> </tr> </table>						1	2	3	4	5	6	1	2	3	4	5	6
1	2	3	4	5	6														
1	2	3	4	5	6														

จากรูปด้านบน แสดงการทำงานของ Binary Search ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

Program Example 17.10: *Binary Search*

```

1 def binarySearch(data, key):
2     min = 0
3     max = len(data) - 1
4
5     while True:

```

```

6         if max < min:
7             return -1
8         m = (min + max) // 2
9         if data[m] < key:
10            min = m + 1
11        elif data[m] > key:
12            max = m - 1
13        else:
14            return m + 1
15
16 data = [1, 2, 3, 4, 5, 6, 7, 8, 9]
17 print("Data source = ", data)
18 print("Would like to search the 8 number.")
19 position = binarySearch(data, 8)
20 print("The position of the 8 number is ", position)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

Data source = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Would like to search the 8 number.
The position of the 8 number is 8

```

การทำงานของการค้นหาข้อมูลแบบนี้ จะทำการรับค่าข้อมูล (Key) สำหรับค้นหาแล้วนำไปค้นหา กับข้อมูลที่เก็บอยู่ในลักษณะพับครึ่ง ทำให้ประสิทธิภาพดีกว่าแบบ Sequential Search สำหรับวิธีการนี้สามารถสรุปประสิทธิภาพในการค้นหาได้เป็น 3 กรณี คือ

1. กรณีที่ดีที่สุด โปรแกรมทำการเปรียบเทียบข้อมูลเพียง 1 ครั้งเท่านั้น ถ้าข้อมูลที่ต้องการค้นหาอยู่ในตำแหน่งที่พับครึ่งแล้วพบในครั้งแรก จะนับค่า BigO = O(1)
2. กรณีที่แย่ที่สุด โปรแกรมทำการเปรียบเทียบข้อมูลจำนวน $\log(n)$ ครั้ง จะนับค่า BigO = $O(\log(n))$
3. กรณีเฉลี่ยทั่วไป โปรแกรมทำการเปรียบเทียบข้อมูลประมาณ $\log(n)$ ครั้ง

จบบทที่ 17

บทที่ 18

ป กิ ณ กะ ป ร แ ก ร ม มิ่ง (Miscellaneous Programming)



ในบทนี้เป็นบทสุดท้ายของภาคที่ 3 ซึ่งเป็นบทที่ผู้เขียนได้รวบรวมเทคนิคการเขียนโปรแกรมเพื่อแก้ปัญหาต่างๆ ด้วยภาษา Python ไว้ หรือพูดอย่างรวมบัด คือ เป็นการเก็บตกหัวข้อที่ไม่ได้กล่าวไว้ในบทที่ 1 ถึง 17 นั่นเอง เนื่องจากเนื้อหาในบทนี้บางส่วนไม่เข้ากับลุमกับเนื้อหาในบทต่างๆ ที่กล่าวมาแล้ว แต่ก็ไม่สามารถตัดทิ้งเนื้อหาในบทนี้ได้เช่นกัน เพราะว่าเป็นเนื้อหาที่มีความสำคัญและมีประโยชน์อย่างมากต่อการเขียนโปรแกรม จึงเรียกชื่อบทนี้ว่า ป กิ ณ กะ ป ร แ ก ร ม มิ่ง (การเขียนโปรแกรมเบ็ดเตล็ด) ซึ่งจะเป็นบทที่ผสมผสานเนื้อหาที่หลากหลายเข้าไว้ด้วยกัน โดยแยกเป็นหัวข้อ ดังนี้

1. วันและเวลา (Date/Time)

Python ใช้โมดูล time เพื่อประมวลผลเกี่ยวกับวันและเวลาของระบบ โดยเวลาเริ่มต้นนับตั้งแต่เวลาเที่ยงคืน ของวันที่ 1 เดือนมกราคม 1970 (Julian day) เป็นต้นมา ซึ่งรูปแบบของวันเวลาจะแสดงอยู่ในรูปแบบของเลขจำนวนจริง (floating-point) มีหน่วยวัดเป็นวินาที เมื่อตัดที่สำคัญของโมดูล time คือ time() เมื่อตัดดังกล่าวจะคืนค่ากลับมาเป็นเลขจำนวนจริงที่เริ่มต้นนับมาตั้งแต่ 12:00am, January 1, 1970 ดังตัวอย่าง

Program Example 18.1: `time()` method

```

⇒1 import time # import the time module
2
⇒3 dateAndTime = time.time()
⇒4 print("Date and Time since 12:00am, January 1, 1970:",
dateAndTime)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

Date and Time since 12:00am, January 1, 1970: 1395544330.215419

จากตัวอย่างโปรแกรมด้านบนแสดงวันและเวลาปัจจุบันของระบบ (Current system time) โดยเวลาดังกล่าวจะมีรูปแบบเป็นเลขจำนวนทศนิยม บรรทัดที่ 1 โปรแกรมนำเข้าโมดูล time เพื่อเรียกใช้งานพังก์ชันหรือเมธอดที่ทำงานเกี่ยวกับเวลาเข้ามาทำงานในโปรแกรม บรรทัดที่ 3 เรียกใช้เมธอด time() ซึ่งจะคืนค่าเวลาปัจจุบันของเครื่องโดยมีรูปแบบเลขทศนิยม ให้กับตัวแปร dataAndTime บรรทัดที่ 4 แสดงผลวันและเวลาดังกล่าวบนจอภาพ

เมธอด time() จะเก็บข้อมูลเป็นชนิดทัพเพิลประกอบไปด้วย 9 พิลเด็คิว

ตำแหน่ง	ฟิลด์	ข้อมูล
0	tm_year: เก็บข้อมูลปี ค.ศ. มี 4 หลัก (Year)	2014
1	tm_mon: เก็บข้อมูลเดือน มกราคม – ธันวาคม (Month)	1 - 12
2	tm_mday: เก็บข้อมูลวันที่ (Day)	1 - 31
3	tm_hour: เก็บข้อมูลชั่วโมง (Hour)	0 - 23
4	tm_min: เก็บข้อมูลนาที (Minute)	0 - 59
5	tm_sec: เก็บข้อมูลวินาที (Second)	0 - 61
6	tm_wday: เก็บข้อมูลวันในสัปดาห์ (วันจันทร์ = 0)	0 - 6
7	tm_yday: เก็บข้อมูลวันใน 1 ปี	1 - 366
8	tm_isdst: เก็บข้อมูล Daylight Savings Time (DST)	-1, 0, 1

สำหรับโมดูล time มีเมธอดที่สำคัญ เช่น time.localtime() สำหรับแสดงข้อมูลวันเวลาปัจจุบันในรูปแบบทัพเพิล, เมธอด time.asctime() แสดงวันเวลาที่ผู้ใช้งานสามารถอ่านได้ง่ายๆ ดังตัวอย่างโปรแกรมที่ 18.2

Program Example 18.2: localtime(), asctime() method

```

1 import time;
2
3 ➔ localtime = time.localtime(time.time())
4 print ("Local current time : ", localtime)
5 ➔ localtime = time.asctime(time.localtime(time.time()))
6 print ("Local current time : ", localtime)

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

Local current time : time.struct_time(tm_year=2014, tm_mon=3,
tm_mday=23, tm_hour=12, tm_min=30, tm_sec=58, tm_wday=6,
tm_yday=82, tm_isdst=0)
Local current time : Sun Mar 23 12:30:58 2014

```

บรรทัดที่ 3 เรียกใช้เมธอด localtime() เพื่อแสดงโครงสร้างของวันเวลา เช่น tm_year=2014 หมายถึงปี ค.ศ. 2014, tm_mon หมายถึงเดือนมีนาคม เป็นต้น บรรทัดที่ 5 เรียกใช้เมธอด asctime() เพื่อแสดงวันเวลาที่ผู้ใช้งานสามารถทำความเข้าใจได้โดยง่าย เช่น Sun Mar 23 12:30:58 2014

ในโมดูล time มีเมธอดที่ถูกใช้งานบ่อยครั้ง ได้แก่ time.ctime() ทำหน้าที่แปลงวันเวลาที่เป็นทศนิยมเป็นสตริงของวันเวลาปัจจุบัน, time.clock() คืนค่าเป็นวินาทีที่ได้จากนาฬิกาของเครื่องคอมพิวเตอร์ และ time.sleep(t) ทำหน้าที่ส่งให้processor หยุดการประมวลผล โดยค่า t มีค่าเท่ากับวินาที

Program Example 18.3: ctime(), clock(), sleep() method

```

1 import time
2
3 t0 = time.clock()
4 print("Start : %s" % time.ctime())
5 print("Start Processing time (seconds) = ", t0)
6 time.sleep(5)
7 print("Stop Processing time (Start - Current time) =", 
8 time.time() - t0)
9 print("End : %s" % time.ctime())

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



OUTPUT

```

Start : Sun Mar 23 12:56:36 2014
Start Processing time (seconds) = 5.131631479069614e-07
Stop Processing time (Start - Current time) =
1395554201.3708725
End : Sun Mar 23 12:56:41 2014

```

จากตัวอย่างโปรแกรมที่ 18.3 แสดงการเรียกใช้งานเมธอด ctime(), clock() และ sleep() บรรทัดที่ 3 โปรแกรมเรียกใช้งาน clock() ซึ่งจะทำให้ได้เวลาจากนาฬิกาของคอมพิวเตอร์มีค่าเป็นเลขทศนิยมเก็บไว้ในตัวแปร t0 บรรทัดที่ 4 สั่งพิมพ์วันเวลาด้วยเมธอด ctime() บรรทัดที่ 5 สั่งพิมพ์เวลาที่เก็บไว้ในตัวแปร t0 ออกทางจอภาพ บรรทัดที่ 6 โปรแกรมสั่งให้โปรแกรมที่กำลังทำงานอยู่หยุดพักการทำงานชั่วคราวประมาณ 5 วินาที ด้วยเมธอด sleep(5) หลังจากโปรแกรมกลับมาทำงานอีกครั้ง จะสั่งพิมพ์เวลาที่หยุดพัก (บรรทัดที่ 7) ไปโดยคำนวณได้จาก เวลาปัจจุบัน (time.time()) – เวลาเริ่มต้น (t0) บรรทัดสุดท้ายโปรแกรมสั่งพิมพ์วันเวลาในรูปแบบสตริงออกทางจอภาพ

2. ส่งจดหมายอิเล็กทรอนิกส์ (Email)

เมื่อผู้เขียนโปรแกรมต้องการส่ง Email ไปยังเพื่อนๆ สามารถทำได้ง่ายๆ โดยใช้โมดูล smtplib ที่ทำงานตามมาตรฐานของโพรโทคอล SMTP (Mail Transfer Protocol) มีรูปแบบการใช้งานคือ

```
smtpObj = smtplib.SMTP([host [, port [, local_hostname]]])
```

โดยพารามิเตอร์ที่ใช้งานมีความหมายดังต่อไปนี้

- host: ระบุชื่อของเซิร์ฟเวอร์ที่ให้บริการเป็นเมล์เซิร์ฟเวอร์ เช่น Hotmail.com, gmail.com และ yahoo.com เป็นต้น (host เป็น option จะไม่ใส่ก็ได้ ถ้าไม่ใส่ข้อมูลให้กับ host เครื่องผู้ใช้งานจำเป็นต้องติดตั้งโปรแกรมที่ทำหน้าที่เป็นเมล์เซิร์ฟเวอร์)

- port: ระบุหมายเลขพอร์ตของเมล์เซิร์ฟเวอร์ โดยปกติจะใช้พอร์ตหมายเลข 25
- local_hostname: เมื่อผู้เขียนโปรแกรมใช้เครื่องที่กำลังทำงานเป็นเมล์เซิร์ฟเวอร์ ให้กำหนดพิลเด็งกล่าวเป็น localhost

เมื่อโปรแกรมสร้างอินสแตนซ์ของคลาส SMTP แล้ว ไฟรอนได้เตรียมเมธอดสำหรับส่งข้อความติดมากับอินสแตนซ์ดังกล่าวด้วยทันที เรียกว่าเมธอด sendmail() โดยมีพารามิเตอร์ 3 ตัวคือ ชื่อผู้ส่ง อีเมล์ (sender), รายชื่อผู้รับอีเมล์ (receivers) สามารถกำหนดได้มากกว่า 1 คน และข้อความที่ต้องการส่ง (message) แสดงในตัวอย่างที่ 18.4

Program Example 18.4: `sendmail()` method

```

⇒1 import smtplib
2
3 sender = 'webmaster@homework.com'
4 receivers = ['abc@gmail.com', 'xyz@hotmail.com']
5
6 message = """From: webmaster <webmaster@homework.com>
To: To every persons <abc@gmail.com>,<xyz@hotmail.com>
Subject: SMTP e-mail test
This is a test e-mail message."""
7
8 try:
9     smtpObj = smtplib.SMTP('localhost')
10    smtpObj.sendmail(sender, receivers, message)
11    print ("Successfully sent email")
12 except SMTPException:
13     print ("Error: unable to send email")

```

ผลลัพธ์ที่เกิดขึ้นจากการรันโปรแกรมดังนี้



Successfully sent email

จากตัวอย่างโปรแกรมที่ 18.4 แสดงวิธีส่งเมล์โดยใช้เมธอด sendmail() เริ่มต้นโปรแกรมนำเข้าโมดูล smtplib ทำหน้าที่เชื่อมต่อกับเมล์เซิร์ฟเวอร์ บรรทัดที่ 3 กำหนดชื่อผู้ส่ง ในที่นี้ทดสอบกำหนดเป็น 'webmaster@homework.com' ซึ่งชื่อดังกล่าวต้องเป็นสมาชิกของผู้ให้บริการเมล์เซิร์ฟเวอร์ด้วย ในการทดสอบนี้ผู้เขียนได้ทำการติดตั้งเมล์เซิร์ฟเวอร์ชื่อ hMailServer บนเครื่อง localhost (ปัจจุบันเมล์เซิร์ฟเวอร์สาธารณะที่ให้บริการเมล์ทั่วๆ ไป เช่น gmail, hotmail ไม่เปิดให้เชื่อมต่อเข้าไปได้โดยตรง) บรรทัดที่ 4 กำหนดรายชื่อผู้รับอีเมล์ซึ่งสามารถมีได้มากกว่า 1 ชื่อ โดยกำหนดไว้ในตัวแปรชนิดลิสต์ บรรทัดที่ 6 สร้างข้อความที่ต้องการส่งอีเมล์ บรรทัดที่ 9 โปรแกรมสร้างอินสแตนซ์ของคลาส SMTP พร้อมกับพารามิเตอร์ 1 ตัวคือ ชื่อของเซิร์ฟเวอร์ที่ให้บริการส่งเมล์ (ในที่นี้ใช้เมล์ที่ติดตั้งอยู่ใน localhost) อีกหนึ่งตัวคือ smtpObj จะถูกนำไปอ้างถึงเมธอด sendmail() เพื่อส่งจดหมาย (บรรทัดที่ 10) โดยเมธอดดังกล่าวต้องการพารามิเตอร์ 3 ตัวคือ ชื่อผู้ส่ง (sender) รายชื่อผู้รับจดหมาย

(receivers) และข้อความที่ต้องการส่ง (message) ถ้าการส่งอีเมล์ทำได้สำเร็จโปรแกรมจะพิมพ์ข้อความว่า "Successfully sent email" แต่ถ้าเกิดข้อผิดพลาดใดๆ เกิดขึ้นในขณะทำการส่งเมล์โปรแกรมจะพิมพ์ข้อความว่า "Error: unable to send email" ดังตัวอย่าง OUTPUT ด้านบน

3. การประยุกต์ใช้ฟังก์ชัน map, filter, reduce, lambda กับข้อมูลแบบลำดับ

ฟังก์ชัน map(): เป็นฟังก์ชันที่ใช้สำหรับดำเนินการกับข้อมูลประเภทลำดับ เช่น list หรือ tuple แต่นิยมใช้กับ list มากที่สุด โดยฟังก์ชัน map() มีหน้าที่จับคู่ฟังก์ชันหรือเมธอดเข้ากับข้อมูลแบบลำดับ เพื่อประหยัดเวลาในการเขียนโปรแกรมลง ซึ่งมีรูปแบบคือ

map(aFunction, aSequence)

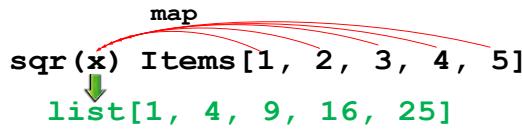
โดย aFunction คือฟังก์ชันใดๆ ที่ต้องการประมวลผล ฟังก์ชันดังกล่าวจำเป็นต้องใช้ข้อมูลใน aSequence เข้ามาทำงานจึงจะสมบูรณ์, aSequence คือลำดับของข้อมูลที่ส่งให้เป็นพารามิเตอร์ของ aFunction ตัวอย่างเช่น ถ้าต้องการเขียนโปรแกรมยกกำลังสองของข้อมูลจำนวนหนึ่ง โดยปกติจะใช้ for ช่วยในการทำงาน เช่น

```
>>> items = [1, 2, 3, 4, 5]
>>> squared = []
>>> for x in items:
...     squared.append(x ** 2)
>>> squared
[1, 4, 9, 16, 25]
```

จากตัวอย่างโปรแกรมข้างต้นจะเห็นว่า โปรแกรมจะดึงข้อมูลในลิสต์มาทีละตัว จากนั้นโปรแกรมจะยกกำลังสองและเพิ่มเข้าไปในตัวแปรชื่อ squared ด้วยเมธอด append() จนกว่าจะครบข้อมูลทุกๆ ตัวใน items แต่ถ้าจะให้สะดวกและประหยัดคำสั่งการเขียนโปรแกรม ให้สังเกตุการใช้ฟังก์ชัน map() ช่วยในการทำงานเดียวกันดังนี้

```
>>> items = [1, 2, 3, 4, 5]
>>> def sqr(x): return x ** 2
>>> lst = list(map(sqr, items))
>>> print(lst)
[1, 4, 9, 16, 25]
```

จากตัวอย่างโปรแกรมข้างบน แสดงให้เห็นว่าฟังก์ชัน map() จะทำการจับคู่ฟังก์ชัน sqr() ซึ่งทำหน้าที่ยกกำลังสองของ x โดย x คือค่าที่ป้อนเข้ามาทีละค่าจากตัวแปร items นั่นเอง ผลลัพธ์ที่ได้คือการยกกำลังสองของสมาชิกทั้งหมดในตัวแปร items ผลลัพธ์ที่ยกกำลังสองแล้วจะเก็บไว้ในตัวแปรชื่อลิสต์เช่นเดิม



ฟังก์ชัน map() ยังสามารถประยุกต์ใช้กับฟังก์ชัน lambda ได้เช่นเดียวกัน ดังนี้

```

>>> lst = list(map(lambda x: x ** 2, items))
>>> print(lst)
[1, 4, 9, 16, 25]
  
```

ฟังก์ชัน lambda ด้านบนทำหน้าที่หาค่ายกกำลังสองของจำนวนเต็มที่อยู่ใน列表 items ได้ เช่น การใช้ฟังก์ชันยกกำลัง pow()

```

>>> import math
>>> pow(3, 5)
243
>>> pow(2, 10)
1024
>>> pow(3, 11)
177147
>>> pow(4, 12)
16777216
>>> list(map(pow, [2, 3, 4], [10, 11, 12]))
[1024, 177147, 16777216]
  
```

จากโปรแกรมข้างบน โปรแกรมจะทำการยกกำลังโดยนำเอาสมาชิกใน list ตัวแรก ([2, 3, 4]) เป็นฐานและนำสมาชิกใน list ตัวที่สอง ([10, 11, 12]) เป็นเลขชี้กำลังในคู่สมาชิกที่ตรงกัน เช่น 2^{10} , 3^{11} , 4^{12} ผลลัพธ์ที่ได้คือ 1024, 177147, 16777216 ตามลำดับ ถ้าพารามิเตอร์ตัวแรกของฟังก์ชัน map() เป็น none ฟังก์ชัน map() จะมองว่าเป็นการสร้างคู่ของข้อมูลชนิดทัพเพิลแทน เช่น

```

>>> m = [1, 2, 3]
>>> n = [1, 4, 9]
>>> new_tuple = map(None, m, n)
>>> new_tuple
[(1, 1), (2, 4), (3, 9)]
  
```

$m = [1, 2, 3]$ $n = [1, 4, 9]$ new_tuple = [(1,1), (2,4), (3,9)]
--

ฟังก์ชัน filter(): เป็นฟังก์ชันที่ใช้สำหรับกรองสมาชิกในลิสต์หรือข้อมูลชนิดลำดับ โดยมีรูปแบบดังนี้

```
filter(aFunction, aSequence)
```

โดย aFunction คือฟังก์ชันใดๆ และ aSequence คือลำดับของข้อมูลที่ต้องการประมวลผลกับ aFunction ตัวอย่างเช่น

```
>>> list(range(-5,5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>> list(filter((lambda x: x < 0), range(-5, 5)))
[-5, -4, -3, -2, -1]
```

จากตัวอย่างโปรแกรมข้างบน ผลลัพธ์ที่เกิดขึ้นจากฟังก์ชัน range(-5, 5) จะได้ช่วงข้อมูลตั้งแต่ -5 ถึง 4 เมื่อต้องการกรองข้อมูลจากผลลัพธ์ที่เกิดขึ้น โดยเอาเฉพาะค่าที่เป็นจำนวนเต็มลบ ผู้เขียนโปรแกรมสามารถใช้ฟังก์ชัน filter() ในการกรองข้อมูลโดยใช้ (lambda x: x < 0) หมายถึง ค่า x ที่เป็นพารามิเตอร์ต้องมีค่าน้อยกว่า 0 เท่านั้น ค่าอื่นๆ จะตัดทิ้งทั้งหมด เมื่อเขียนโปรแกรมด้วยลูป for แบบธรรมดาก็จะให้ผลลัพธ์ที่เหมือนกับโปรแกรมด้านบนแสดงได้ดังนี้

```
>>> result = []
>>> for x in range(-5, 5):
    if x < 0:
        result.append(x)
>>> result
[-5, -4, -3, -2, -1]
```

ฟังก์ชัน reduce(): เป็นฟังก์ชันที่ใช้สำหรับลดการประมวลผลของกระบวนการลูปกับข้อมูลแบบลำดับ โดยมีรูปแบบดังนี้

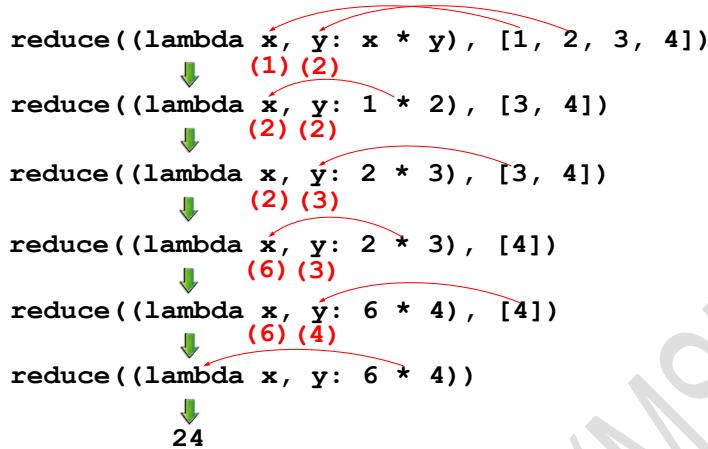
reduce(aFunction, aSequence)

โดย aFunction คือฟังก์ชันใดๆ และ aSequence คือลำดับของข้อมูลที่ต้องการประมวลผลกับ aFunction แต่ก่อนการใช้งานจำเป็นต้องนำเข้าโมดูล functools เข้ามา ก่อน ตัวอย่างเช่น

```
>>> from functools import reduce
>>> reduce((lambda x, y: x * y), [1, 2, 3, 4])
24
>>> reduce((lambda x, y: x / y), [1, 2, 3, 4])
0.04166666666666664
```

จากตัวอย่างโปรแกรมข้างบนเป็นการการทำงานของฟังก์ชัน reduce() โดยบรรทัดที่ 1 เป็นการนำเข้าฟังก์ชัน reduce() ที่อยู่ในโมดูล functools เข้ามาทำงาน บรรทัดต่อมาเรียกใช้ฟังก์ชัน (lambda x, y: x * y) ซึ่งหมายถึง การหาผลคูณของ x กับ y โดยในรอบแรกฟังก์ชัน lambda นำเอาสมาชิกตัวที่ 1 (1) ในลิสต์ส่งเป็นพารามิเตอร์ให้กับตัวแปร x และสมาชิกตัวที่ 2 (2) ส่งเป็นพารามิเตอร์ให้กับตัวแปร y นำมาคูณกัน ผลลัพธ์ที่ได้จากการคูณซึ่งมีค่าเท่ากับ 2 ($1*2=2$) จะกล้ายเป็นพารามิเตอร์ของ x อีกครั้ง และ lambda ก็จะดึงสมาชิกตัวที่ 3 (3) ในลิสต์ ให้เป็นพารามิเตอร์ y จากนั้น lambda จะนำ x และ y มาคูณกันอีกครั้ง ผลลัพธ์ที่ได้เท่ากับ 6 ($2*3=6$) เก็บไว้ในตัวแปร x และนำไปคูณกับตัวแปร y ซึ่งมีค่า

เท่ากับ 4 อีกรึ ผลลัพธ์ที่ได้เท่ากับ 24 ($6*4=24$) การทำงานของฟังก์ชัน reduce() ทำงานค่อนข้างซับซ้อน ดังนั้นจำเป็นต้องดูภาพด้านล่างเพื่อประกอบความเข้าใจด้วย



สำหรับฟังก์ชัน (lambda x, y: x / y) จะทำงานคล้ายกับ lambda การคูณที่แสดงในรูปด้านบน แต่แตกต่างกันคือ เปลี่ยนจากการคูณเป็นการหารแทน ผลลัพธ์ที่ได้เท่ากับ 0.04166 เมื่อเขียนโปรแกรมโดยใช้ลูปธรรมดาเพื่อประมวลผลการคูณเหมือนฟังก์ชัน reduce() แสดงได้ดังนี้

```

>>> L = [1, 2, 3, 4]
>>> result = L[0]
>>> for x in L[1:]:
    result = result * x
>>> result
24

```

ฟังก์ชัน lambda(): ทำหน้าที่แทนการสร้างฟังก์ชันแบบปกติ โดยไม่จำเป็นต้องประกาศชื่อฟังก์ชันแต่ใช้คำว่า lambda แทน ใช้สำหรับประมวลผลเพื่อแก็บัญญาติอย่างเหมือนกับการสร้างสูตรหรือสมการในตารางของโปรแกรม MS-Excel (เนื้อหาโดยละเอียดสามารถอ่านได้จากบทที่ 7) โดยมีรูปแบบดังนี้

lambda para1, para2,...,paraN: operation

โดย lambda คือชื่อของฟังก์ชัน, para1 – paraN คือพารามิเตอร์ที่ส่งให้กับ lambda และ operation คือคำสั่งที่ lambda ต้องประมวลผล เช่น

ถ้าต้องการหาค่ายกกำลัง 3 ของ x สามารถเขียนโปรแกรมด้วยฟังก์ชันปกติได้คือ

```

>>> def func(x): return x ** 3
>>> print(func(5))
125

```

แต่ถ้าใช้ฟังก์ชัน lambda โปรแกรมจะสั้นและกระหัดกระหั่นกว่าโดยไม่ต้องใช้คำสั่ง return ดังนี้

```
>>> lamb = lambda x: x ** 3
>>> print(lamb(5))
125
```

4. ไฟรอน Generators, yield

Generators เป็นกลไกพิเศษที่ใช้สำหรับควบคุมพฤติกรรมการทำงานของลูปเพื่อสร้างชุดของข้อมูลให้เป็นไปตามที่ผู้ใช้ต้องการ ซึ่งในความเป็นจริงแล้ว Generator มีลักษณะการทำงานที่คล้ายคลึงกับฟังก์ชันเป็นอย่างมาก แตกต่างกันคือ ฟังก์ชันจะคืนค่ากลับให้แก่ผู้เรียกใช้งานทั้งหมดในครั้งเดียว เมื่อทำงานเสร็จ (อาจจะเป็นข้อมูลเพียงค่าเดียวหรือเป็นชุดของข้อมูลก็ได้) ด้วยคำสั่ง return แต่สำหรับ Generators และข้อมูลที่ส่งคืนให้แก่ผู้เรียกจะขึ้นอยู่กับความต้องการใช้งาน (On-demand) กล่าวคือ ถ้าผู้ใช้ต้องการ Generators จะสร้างข้อมูลให้ตามที่ต้องการเท่านั้น หลังจากนั้น Generators จะหยุดพักทำงานชั่วคราว (Suspending) และจะกลับมาทำงานใหม่อีกรอบเมื่อผู้ใช้งานเรียกใช้งานอีกรอบ (Resuming) ทำอย่างนี้ไปเรื่อยๆ จนกว่าผู้ใช้จะยุติการใช้งาน ซึ่งวิธีการแบบ Generators นี้จะช่วยให้คอมพิวเตอร์ประหยัดหน่วยความจำ ซึ่งแตกต่างจากฟังก์ชันที่จะสร้างชุดข้อมูลที่มีค่าคงที่ เช่น อาร์เรย์ ลิสต์ หรือทัพเพิล เพื่อส่งคืนให้แก่ผู้ใช้งาน Generators แบ่งออกเป็น 2 ประเภทคือ

- Generator functions คือ การใช้คำสั่ง yield ทำงานร่วมกับฟังก์ชันปกติ โดยข้อมูลที่ถูกสร้างจะขึ้นอยู่กับความต้องการของผู้ใช้งานขณะนั้นๆ โดยปราศจากคำสั่ง return เมื่อผู้ใช้ยังไม่เรียกใช้งาน ฟังก์ชันดังกล่าวจะหยุดพัก (Suspending) และกลับมาทำงานใหม่เมื่อผู้ใช้เรียกใช้งาน (Resuming) อีกรอบ ทำอย่างนี้ไปเรื่อยจนกว่าจะยุติโปรแกรม
- Generator expressions เป็นการฝังนิพจน์คณิตศาสตร์หรือคำสั่งลงในตัวแปรประเภทลิสต์ เพื่อสร้างชุดของข้อมูล เช่น

```
>>> list(c * 3 for c in 'Python')
['PPP', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
>>> list(x ** 3 for x in range(5))
[0, 1, 8, 27, 64]
```

Generator functions

จากที่กล่าวมาแล้วข้างต้นว่า Generator functions สร้างขึ้นจากคำสั่ง yield ร่วมกับฟังก์ชันปกติ โดยไม่มีคำสั่ง return ปิดท้ายฟังก์ชัน จากตัวอย่างข้างล่าง แสดงการสร้างชุดข้อมูลเลขจำนวนนับที่ยกกำลังสอง

```
def increment():
    x = 0
```

```

while True:
    x += 1
    yield x

y = increment()
print(y)
print(next(y))
print(next(y))
print(next(y))

```

ผลลัพธ์ที่ได้จากโปรแกรมคือ



OUTPUT

```
<generator object increment at 0x000000004008828>
1
2
3
```

จากตัวอย่างโปรแกรมข้างบน โปรแกรมสร้างฟังก์ชันชื่อ `increment()` ทำหน้าที่เพิ่มค่าข้อมูลขึ้นครั้งละ 1 เมื่อเรียกใช้งานฟังก์ชันดังกล่าว การเรียกใช้คำสั่ง `y = increment()` จะส่งผลให้ค่าที่คืนกลับมาจากการเรียกใช้งานฟังก์ชันดังกล่าวคือ ที่อยู่ของอ็อปเจกต์ Generator (`<generator object increment at 0x000000004008828>`) ซึ่งไม่ใช้ข้อมูลที่สร้างมาจากฟังก์ชันดังกล่าว ผู้ใช้งานจำเป็นต้องใช้ฟังก์ชัน `next(y)` เพื่อช่วยดึงข้อมูลจากฟังก์ชัน `increment()` มาแสดงผลทีละค่า แต่ถ้าต้องการเรียกใช้งานข้อมูลทั้งหมดอย่างต่อเนื่องสามารถใช้งานร่วมกับคำสั่งลูป `for` ได้ เช่น

Program Example 18.5: `yield, next`

⇒1	<code>def increment():</code>
2	<code>x = 0</code>
3	<code>while True:</code>
⇒4	<code>yield x</code>
⇒5	<code>x += 1</code>
6	
⇒7	<code>def pow(n):</code>
8	<code>for i in range(n):</code>
9	<code>yield i**2</code>
10	
11	<code>y = increment()</code>
⇒12	<code>for j in pow(6):</code>
⇒13	<code>print("The power of %d = %d" %(next(y), j))</code>

ผลลัพธ์ที่ได้จากโปรแกรมคือ



OUTPUT

```
The power of 0 = 0
The power of 1 = 1
The power of 2 = 4
The power of 3 = 9
The power of 4 = 16
The power of 5 = 25
```

จากการทำงานของโปรแกรมที่ 18.5 บรรทัดที่ 4 ในฟังก์ชัน `increment()` สังเกตว่า คำสั่ง `yield` แตกต่างจากคำสั่ง `return` เพราะถ้าใช้คำสั่ง `return` แทน `yield` แล้ว คำสั่ง `x += 1` จะไม่ถูกทำงานเลย (เพราะจะกลับไปยังโปรแกรมที่เรียกใช้งานทันที) แต่สำหรับคำสั่ง `yield` คำสั่ง `x += 1` จะสามารถทำงานได้

หลังจากพิมพ์ชั้น increment() กลับมาทำงานอีกรัง บรรทัดที่ 7 เมื่อ dot pow() สร้างชุดข้อมูลที่เป็นเลขจำนวนนับยกกำลังสองร่วมกับคำสั่ง yield โดยรับพารามิเตอร์ 1 ตัว คือ เลขจำนวนนับ (n) บรรทัดที่ 12 และ 13 โปรแกรมสั่งพิมพ์ค่าเลขยกกำลังสองออกจอภาพด้วยพิมพ์ print() ทำงานร่วมกับลูป for โดยใช้พารามิเตอร์ 2 ตัวคือ next(y) ซึ่งเป็นชุดข้อมูลที่เกิดจากพิมพ์ชั้น increment() และ j ซึ่งเป็นชุดข้อมูลที่เกิดจากพิมพ์ชั้น power()



Note: ผู้เขียนโปรแกรมไม่สามารถใช้คำสั่ง return ในการสร้างชุดข้อมูลได้เหมือนคำสั่ง yield เช่น def generator(n):

```
for i in range(n):
    return i ** 3

for i in generator(5):
    print(i)
```

TypeError: 'int' object is not iterable

5. การเขียนโปรแกรมไฟล์นัดด้วย Eclipse

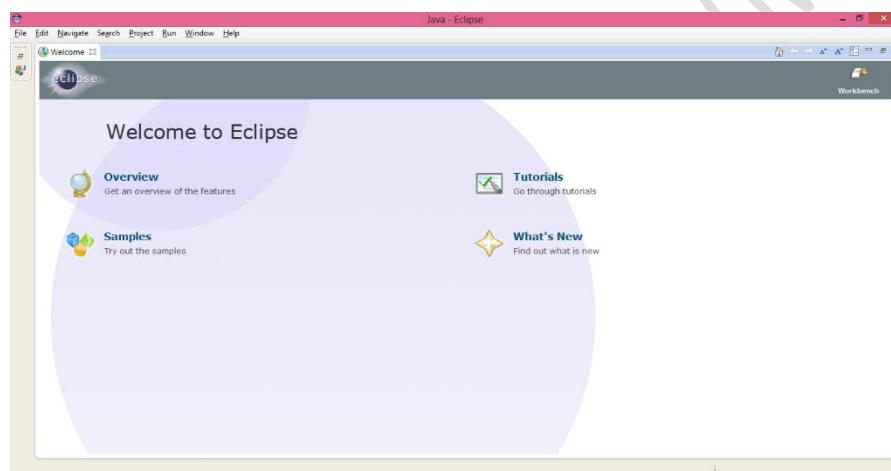
Eclipse เป็นเครื่องมือที่ได้รับความนิยมเป็นอย่างมากสำหรับโปรแกรมเมอร์เพื่อใช้พัฒนาโปรแกรมได้หลากหลาย เช่น Java, C, C++ เป็นต้น เนื่องจากโปรแกรมดังกล่าวมีรูปแบบอินเตอร์เฟสที่สามารถใช้งานได้สะดวก ในหัวข้อนี้ผู้เขียนจะแนะนำการใช้งานโปรแกรมดังกล่าว โดยมีขั้นตอนดังนี้

- 1) ดาวน์โหลดและติดตั้ง Java JRE หรือ Java SDK ก่อน จาก URL:
<https://www.java.com/en/download/> หรือจาก CD ซึ่งอยู่ในไดเรคทรอรี่ Ch18-Misc\Softwares\jre-7u51-windows-x64.exe เมื่อติดตั้งเสร็จแล้วต้องกำหนด path ใน Environment Variables ในวินโดวส์ให้ไปยัง C:\Program Files (x86)\Java\jre7\bin
- 2) ดาวน์โหลดโปรแกรม Eclipase มาติดตั้งบนเครื่องก่อนจาก URL:
<https://www.eclipse.org/downloads/> โดยเลือกแบบ Eclipse Standard x.x.x (x คือ หมายเลขของเวอร์ชัน) ซึ่งมีขนาดประมาณ 200 Mb หรือสามารถติดตั้งโปรแกรมได้จากแฟ้ม CD ซึ่งอยู่ในไดเรคทรอรี่ Ch18-Misc\Softwares\eclipse-standard-kepler-SR2-win32-x86_64.zip สำหรับสถาปัตยกรรมแบบ 64 บิต เมื่อดาวน์โหลดมาแล้วให้ทำการแตกแฟ้มที่บีบอัดไว้ แล้วนำไปวางไว้ในไดรฟ์ C:
- 3) ดับเบิลคลิกแฟ้มชื่อ eclipse.exe ดังรูปที่ 1



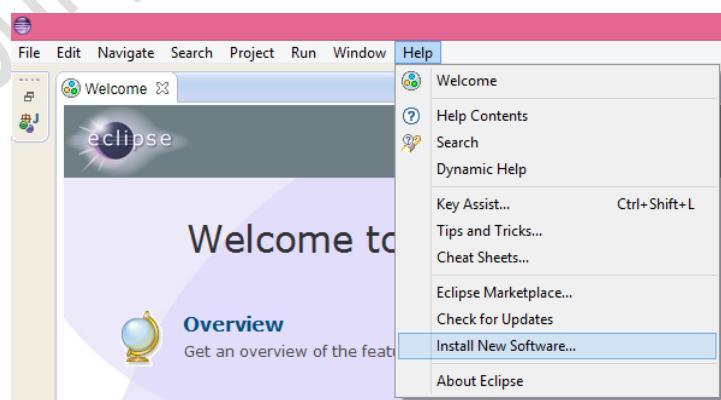
รูปที่ 1 แสดงขั้นตอนเริ่มต้นการปรับแต่ง eclipse

- 4) คลิกเลือก OK จากนั้นโปรแกรม eclipse จะเริ่มต้นทำงานดังรูปที่ 2



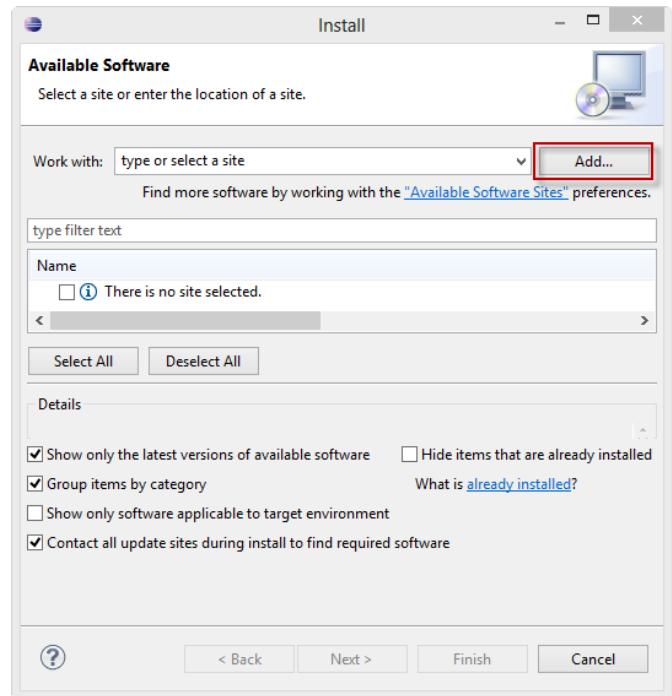
รูปที่ 2 แสดงหน้าต่างเริ่มต้นของโปรแกรม eclipse

- 5) เลือกที่เมนู Help \Rightarrow Install New Software... ดังรูปที่ 3



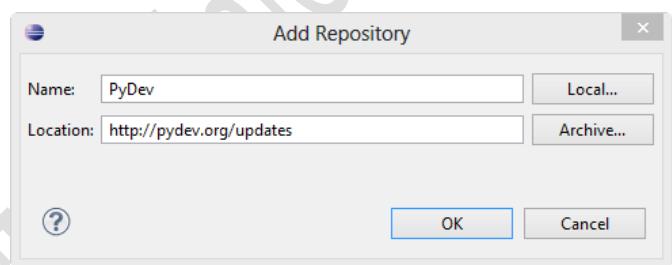
รูปที่ 3 แสดงการติดตั้งซอฟต์แวร์เพิ่มเติม

- 6) คลิกเลือก Add... ดังรูปที่ 4



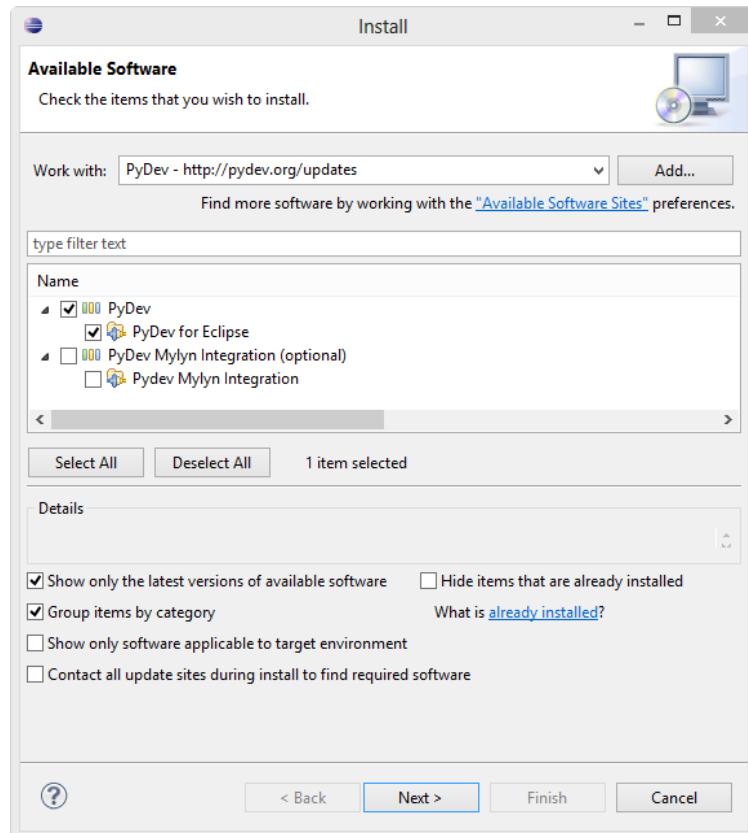
รูปที่ 4 เพิ่มแหล่งที่เก็บตัวคอมไพล์ภาษาไทยบนให้กับ eclipse

- 7) ให้เพิ่ม URL ของเว็บไซต์ pydev.org ในช่อง Name: เพิ่มข้อความว่า PyDev และช่อง Location: เป็น http://pydev.org/updates เพื่อทำการเพิ่มภาษาไทยบนให้กับ eclipse ดังรูปที่ 5 ต่อจากนั้กดปุ่ม OK



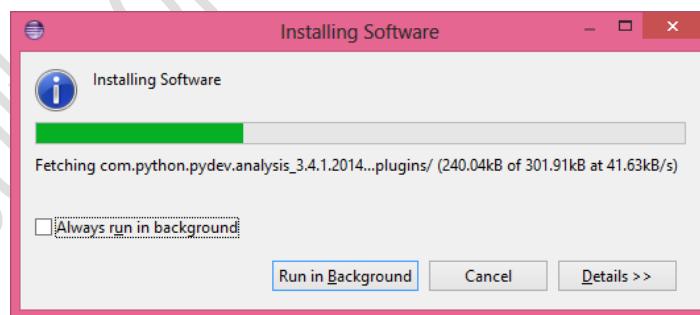
รูปที่ 5 เพิ่มแหล่งข้อมูลภาษาไทยบนจากเว็บไซต์ pydev.org

- 8) รอสักครู่ โปรแกรมจะทำการ update ข้อมูลให้กับ eclipse เมื่อโปรแกรมปรับปรุงข้อมูลสำเร็จ โปรแกรมจะแสดงดังรูปที่ 6 ให้คลิกเลือก PyDev และ PyDev for Eclipse และเลือก Next>



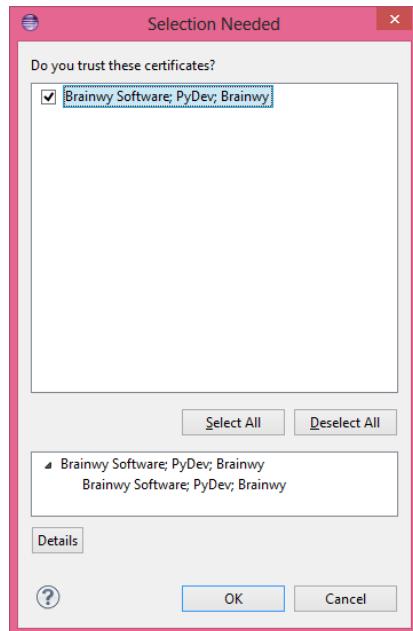
รูปที่ 6 คลิกเลือก PyDev for Eclipse

- 9) ขั้นตอนต่อไปเลือก Next> เมื่อถึงเมนู Review Licenses เลือก I accept the terms.... และคลิกเลือก Finish
- 10) โปรแกรมจะเริ่มทำการติดตั้งโปรแกรมเสริมคือ ไฟรอนให้กับ eclipse ให้รอสักครู่ ดังรูปที่ 7



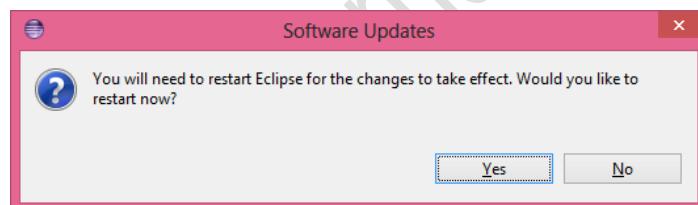
รูปที่ 7 โปรแกรมเริ่มติดตั้งไฟรอนสำหรับ eclipse

- 11) เมื่อโปรแกรมแสดงเมนู Selection Needed ให้คลิกเลือก Brainwy Software; PyDev;Brainwy และเลือก OK ดังรูปที่ 8



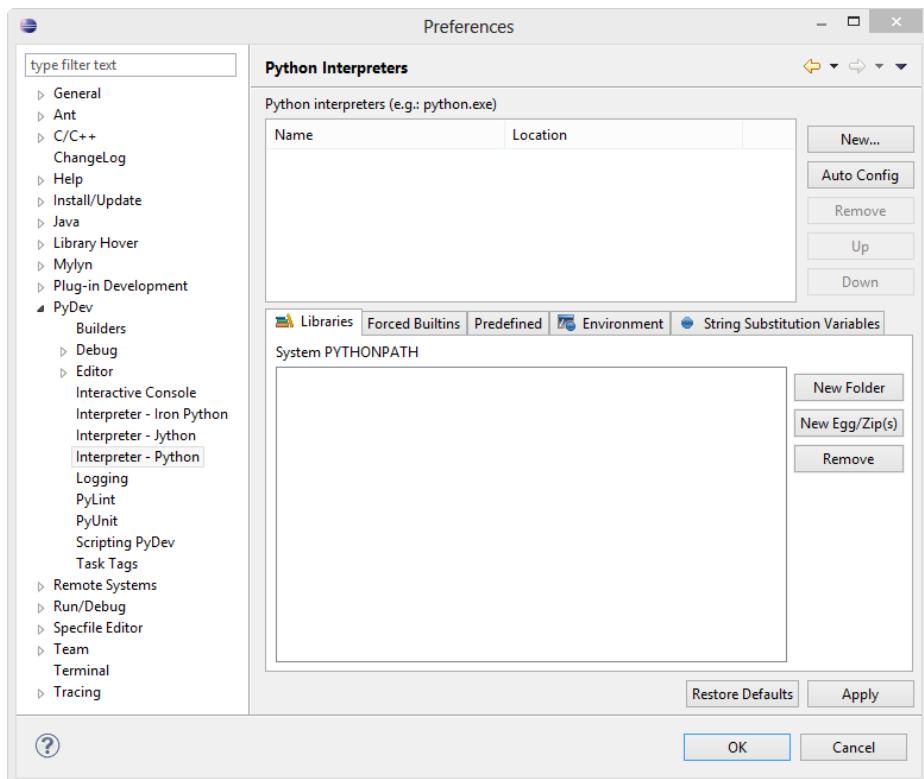
รูปที่ 8 แสดงหน้าต่าง Selection Needed

- 12) เมื่อติดตั้งเสร็จ โปรแกรมแจ้งให้ผู้ใช้ทราบว่าจำเป็นต้องเริ่มต้นการทำงานของ eclipse ใหม่ ให้ผู้ใช้เลือก Yes ดังรูปที่ 9



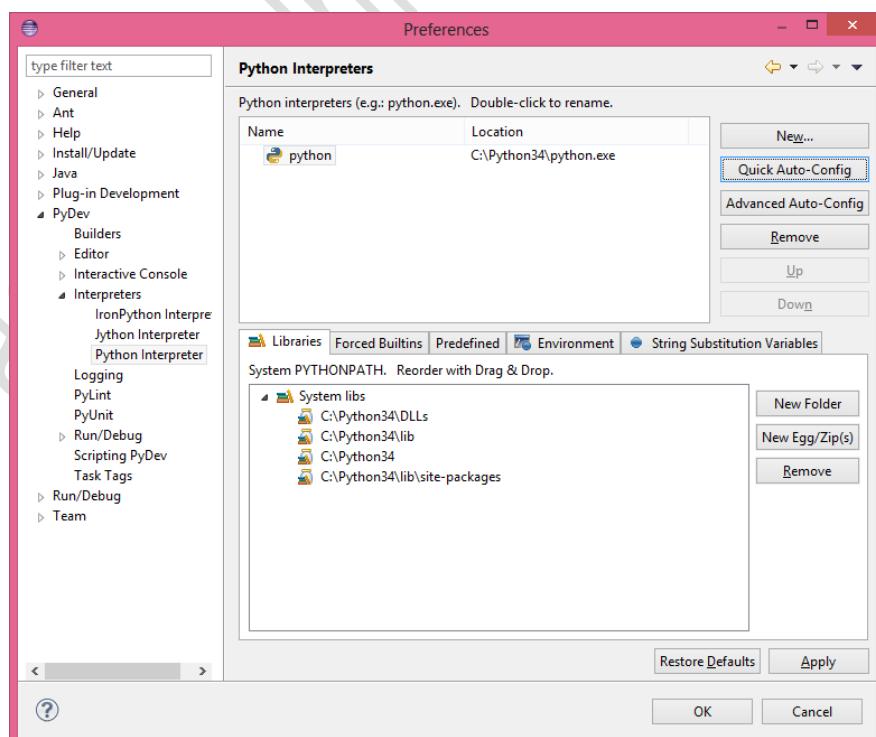
รูปที่ 9 แสดงโปรแกรมติดตั้งเสร็จและต้องการเริ่มต้นโปรแกรมใหม่

- 13) เมื่อเริ่มต้นโปรแกรมใหม่ จำเป็นต้องปรับแต่งให้ eclipse มองเห็นตัวแปรภาษา Python ก่อน เริ่มต้นใช้งาน eclipse คลิกที่ Window \Rightarrow Preferences \Rightarrow PyDev \Rightarrow Interpreter \Rightarrow Interpreter - Python ดังรูปที่ 10



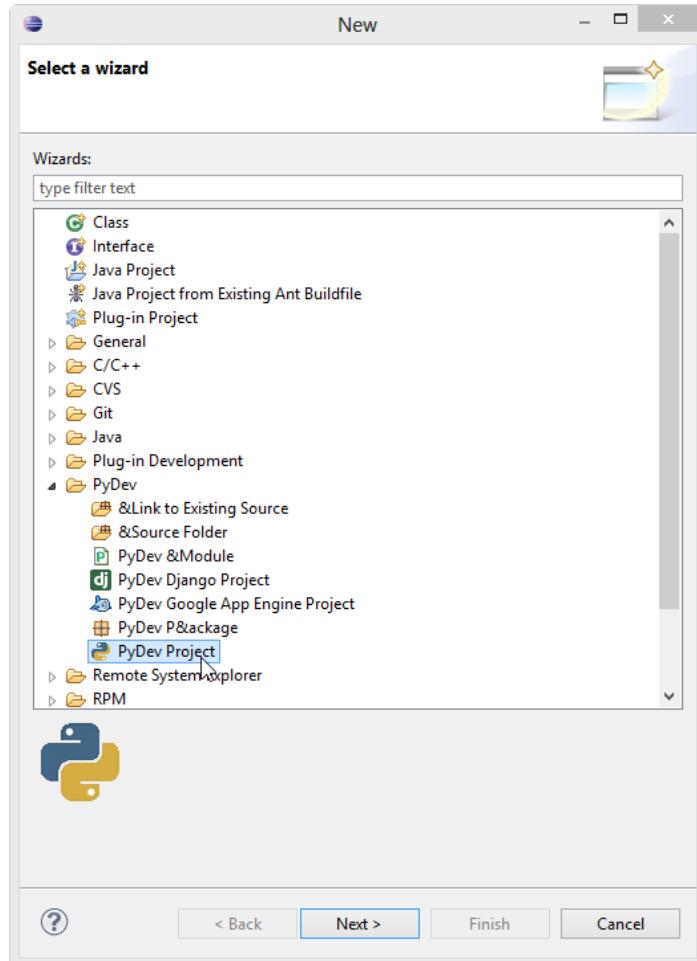
รูปที่ 10 แสดงหน้าต่าง Interpreter - Python

- 14) ในหน้าต่าง Python Interpreters เลือก Quick Auto-Config ซึ่งจะทำให้โปรแกรมค้นหาตำแหน่งที่อยู่ของไฟร์อนให้เองโดยอัตโนมัติ แล้วคลิก OK ดังรูปที่ 11



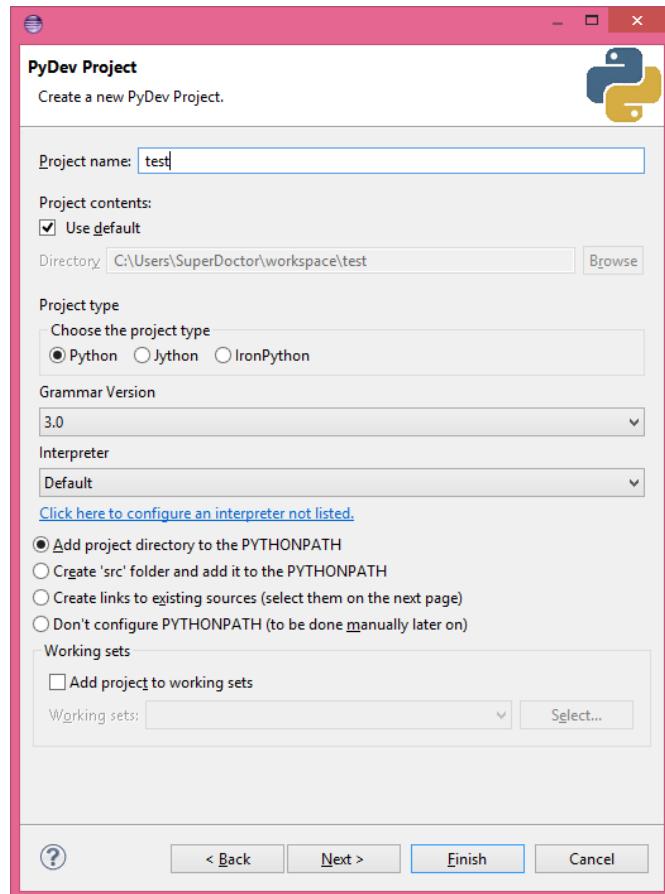
รูปที่ 11 แสดงผลลัพธ์เมื่อเลือก Quick Auto-Config

- 15) เมื่อ eclipse ได้รับการปรับแต่งเรียบร้อยแล้ว ขั้นตอนต่อไปจะเป็นการสร้าง Project เพื่อใช้สำหรับเขียนโปรแกรม โดยเลือกที่ File \Rightarrow New \Rightarrow Other... และเลือกที่ PyDev Project แล้วคลิก Next > ดังรูปที่ 12



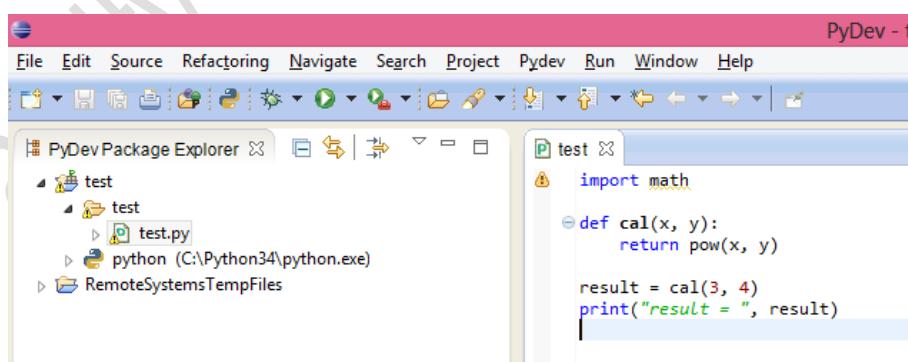
รูปที่ 12 แสดงการสร้าง Project

- 16) ทดสอบตั้งชื่อ Project ชื่อ test ชื่อ Project ดังกล่าวจะบันทึกอยู่ในไดเรคทรอรี่ C:\Users\Username\workspace ต่อจากนั้นให้เลือก Grammar version เป็น 3.0 และเลือก Finish ดังรูปที่ 13



รูปที่ 13 เสร็จสิ้นขั้นตอนการสร้าง Project

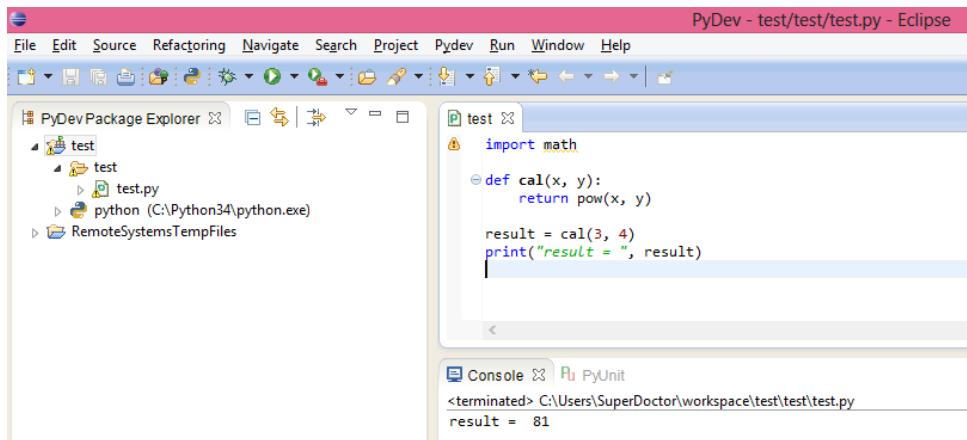
- 17) ทดสอบการรันโปรแกรม โดยเลือกที่ Test Project \Rightarrow New \Rightarrow Source Folder ตั้งชื่อว่า test จากนั้นคลิกขวาที่ Source Folder ชื่อว่า test และเลือก New \Rightarrow File \Rightarrow ตั้งชื่อเพิ่มว่า test.py และทดสอบเขียนโปรแกรมดังรูปที่ 14



รูปที่ 14 สร้างไดเรกทอรีสำหรับเก็บโปรแกรม

- 18) เมื่อสั่งรันโปรแกรม โดยการกดที่เมนู Run \Rightarrow Run หรือปุ่ม CTRL + 11 ถ้าโปรแกรมแสดงข้อความผิดพลาดว่า "Fatal Python error: Py_Initialize: can't initialize sys standard streams LookupError: unknown encoding: MS874" แสดงว่าfonต์ที่

โปรแกรมกำหนดไว้ วินโดวส์ไม่สนับสนุน ให้ทำการกำหนดฟอนต์ใหม่ โดยการคลิกขวาที่ชื่อ Project (ในที่นี้คือ Test) \Rightarrow Properties \Rightarrow Resource \Rightarrow Text file encoding \Rightarrow UTF-8 \Rightarrow คลิก OK และสั่งรันโปรแกรมใหม่อีกครั้ง ถ้าไม่มีข้อผิดพลาดใดๆ ผลลัพธ์จะแสดงดังรูปที่ 15



```
PyDev - test/test/test.py - Eclipse
File Edit Source Refactoring Navigate Search Project Pydev Run Window Help
PyDev Package Explorer test
  test
    test.py
    python (C:\Python34\python.exe)
  RemoteSystemsTempFiles
test
  import math
  def cal(x, y):
      return pow(x, y)

  result = cal(3, 4)
  print("result = ", result)

Console PyUnit
<terminated> C:\Users\SuperDoctor\workspace\test\test\test.py
result = 81
```

รูปที่ 15 แสดงการรันโปรแกรม



Caution! เมื่อสั่งรันโปรแกรมไฟรอนบน eclipse และเกิดข้อผิดพลาดคือ "Fatal Python error: Py_Initialize: can't initialize sys standard streams LookupError: unknown encoding: MS874" ให้แก้ไขโดย คลิกขวาที่ชื่อ Project \Rightarrow Properties \Rightarrow Resource \Rightarrow Text file encoding \Rightarrow UTF-8 \Rightarrow คลิก OK

6. การแปลงไฟรอนสคริปต์เป็น EXE (exeutable)

ไฟรอนเป็นภาษาสคริปต์ ดังนั้นเมื่อรันโปรแกรมจำเป็นต้องมีตัวแปรภาษาอยู่ในเครื่องที่ทำงานเสมอ และมีข้อจำกัดเรื่องความเร็วในการทำงานด้วย เพราะต้องแบลกภาษาที่ลับบรรทัด (Interpreting) จากบัญหาดังกล่าวสามารถแก้บัญหาโดยการแปลงภาษาสคริปต์เป็นแฟ้มประเภท exe เพื่อให้สามารถประมวลผลได้เร็วขึ้นและสามารถทำงานได้โดยปราศจากตัวแปรภาษา ซึ่งมีขั้นตอนดังนี้

- 1) บนเครื่องคอมพิวเตอร์ต้องมีการติดตั้งภาษาไฟรอน ในตัวอย่างนี้เป็นเวอร์ชัน 3.4
- 2) ต้องติดตั้งโปรแกรม py2exe สามารถดาวน์โหลดได้จาก URL: <http://www.py2exe.org/> สำหรับไฟรอน 2.7 และ cx_Freeze สำหรับไฟรอนเวอร์ชัน 2.7 ขึ้นไป ดาวน์โหลดได้จาก URL: <http://cx-freeze.sourceforge.net/> (โปรแกรมทั้งหมดอยู่ในแผ่น CD ที่แนบมา กับหนังสือเล่มนี้แล้วในโฟลเดอร์ Ch18\Softwares)

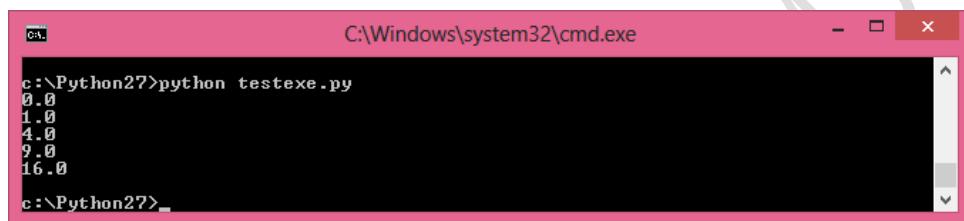
1. การแปลงสคริปต์ไฟรอนเป็น exe ด้วย py2exe

สำหรับโปรแกรม py2exe จะเหมาะกับผู้เขียนโปรแกรมที่ยังใช้แพลตฟอร์มตั้งแต่ 2.7 ลงไป ซึ่งมีขั้นตอนดังนี้

- หลังจากติดตั้งโปรแกรม py2exe เสร็จเรียบร้อยแล้วให้เขียนโปรแกรมทดสอบว่ามีชื่อว่า testexe.py ดังนี้

```
import math
def cal(n):
    for i in range(n):
        print(math.pow(i, 2))
cal(5)
```

- ทดสอบรันโปรแกรมผ่านทาง MS-DOS ด้วยคำสั่ง python testexe.py ดังรูปด้านล่าง

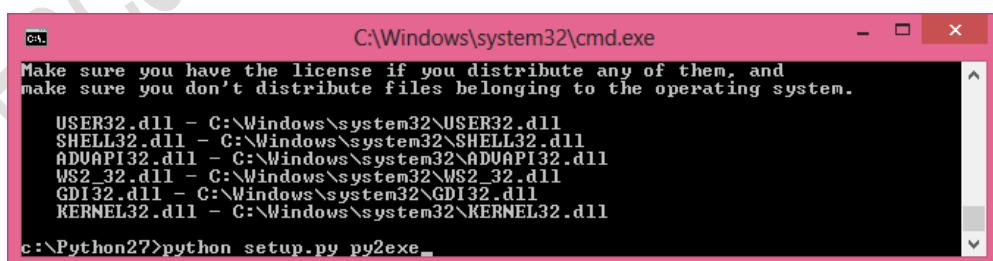


```
c:\Python27>python testexe.py
0.0
1.0
4.0
9.0
16.0
c:\Python27>
```

- อันดับต่อไป ให้ทำการสร้างแฟ้มสำหรับกำหนดสภาพแวดล้อมในการสร้างแฟ้ม exe ชื่อว่า setup.py ซึ่งมีคำสั่งดังนี้

```
from distutils.core import setup
import py2exe
setup(console=['testexe.py'])
```

จากโปรแกรม setup.py ทำการนำเข้าคลาส setup จากโมดูล distutils.core เข้ามาทำงานเพื่อรวมรวมไลบรารีต่างๆ ที่จำเป็นสำหรับการสร้างแฟ้ม exe บรรทัดต่อมานำเข้าโปรแกรม py2exe เพื่อสร้างแฟ้ม exe บรรทัดสุดท้ายกำหนดชื่อแฟ้มไฟล์ในสคริปต์ที่ต้องการแปลงเป็นแฟ้ม exe ไว้ภายในเครื่องหมาย [...] เมื่อสร้างแฟ้มดังกล่าวเสร็จแล้ว ให้สร้างแฟ้ม exe โดยใช้คำสั่ง **python setup.py py2exe** ใน MS-DOS ผลลัพธ์แสดงดังภาพด้านล่าง



```
Make sure you have the license if you distribute any of them, and
make sure you don't distribute files belonging to the operating system.

USER32.dll - C:\Windows\system32\USER32.dll
SHELL32.dll - C:\Windows\system32\SHELL32.dll
ADVAPI32.dll - C:\Windows\system32\ADVAPI32.dll
WS2_32.dll - C:\Windows\system32\WS2_32.dll
GDI32.dll - C:\Windows\system32\GDI32.dll
KERNEL32.dll - C:\Windows\system32\KERNEL32.dll

c:\Python27>python setup.py py2exe
```



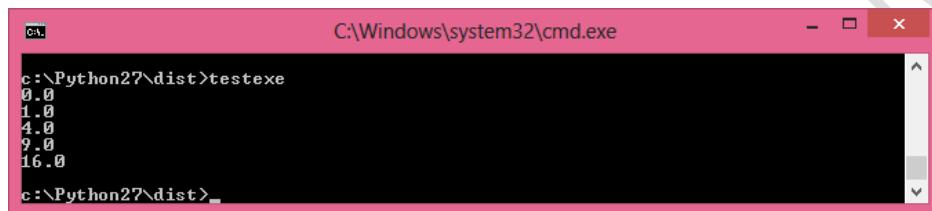
Caution! ถ้าสั่งคอมpileโปรแกรมเป็น exe และเกิดข้อผิดพลาด คือ Traceback (most recent call last):

File "setup.py", line 2, in <module>

import py2exe

ImportError: No module named 'py2exe' แสดงว่าไม่ได้ทำการ path โปรแกรม py2exe ไว้ดังนั้นให้ย้ายแฟ้ม setup.py และ testexe.py ไว้ใน path ของไฟรอน เช่น C:\Python27 แทน

- ผลลัพธ์จากการคอมไพล์จะสร้างแฟ้ม exe และแฟ้มข้อมูลที่เกี่ยวข้อง เช่น .dll, library.zip, .pyd เก็บไว้ในไฟเด rektror ชื่อ dist ดังนั้นมือผู้ใช้ต้องการนำแฟ้ม .exe ไปทำงานบนเครื่องอื่นๆ จำเป็นต้องคัดลอกไฟเด rektror ดังกล่าวไปทั้งหมด มิเช่นนั้นแล้วจะเกิดข้อผิดพลาดทันที ให้ผู้ใช้ทดลองดับเบิลคลิกแฟ้ม testexe.exe หรือส่องรันผ่าน MS-DOS ผลลัพธ์แสดงดังรูปด้านล่าง



2. การแปลงสคริปต์ไฟรอนเป็น exe ด้วย cx_Freeze

- ติดตั้งโปรแกรม cx_Freeze โดยดับเบิลคลิกแฟ้มชื่อ cx_Freeze-4.3.2*.msi ให้ผู้ใช้เลือก Next > ไปเรื่อยๆ จนกว่าโปรแกรมจะเสร็จสิ้นการติดตั้ง
- โปรแกรม cx_Freeze จะทำการ path ให้อัตโนมัติ ผู้ใช้งานสามารถเรียกใช้งานได้ทันที
- การสร้าง exe ด้วย cx_Freeze ทำได้ 2 วิธีคือ วิธีที่ 1 สั่งรันด้วยคำสั่ง **cxfreeze testexe.py --target-dir dist** ใน MS-DOS ซึ่งจะทำให้ได้แฟ้ม .exe ทันทีและเก็บแฟ้มดังกล่าวในไฟเด rektror ชื่อ dist วิธีที่ 2 คือการสร้างแฟ้มชื่อ setup.py ซึ่งมีคำสั่งดังนี้ หรือค้นหาได้จากแฟ้ม CD

```
from cx_Freeze import setup, Executable

setup( name = "hello",
       version = "0.1",
       description = "Sample cx_Freeze script",
       executables = [Executable("testexe.py")])
```

- สั่งคอมไพล์โปรแกรมด้วยคำสั่ง **python setup.py build** ใน MS-DOS ผลจากการคอมไпал์โปรแกรมจะสร้างไฟล์เด rektror ชื่อ built\exe.win-xx และเก็บแฟ้ม .exe ไว้ในไฟล์เด rektror ดังกล่าว

7. Decorator Vs Function

ฟังก์ชัน (Function) คือ กลุ่มของชุดคำสั่งที่ทำงานเฉพาะกิจ เช่น คำนวณเลขยกกำลัง หากค่าพื้นเป็นต้น แต่ Decorator คือ ฟังก์ชันที่ทำหน้าที่ดัดแปลงหรือปรับแต่งฟังก์ชันอื่นๆ ให้เหมาะสมตามที่ผู้เขียนโปรแกรมต้องการ

ไฟรอนมองทุกๆ สิ่งเป็นอ็อปเจกต์ทั้งหมด ไม่มีข้อยกเว้นแม้แต่ฟังก์ชัน ซึ่งส่งผลให้ฟังก์ชันถูกมองว่าเป็นพารามิเตอร์ตัวหนึ่งในโปรแกรมก็ได้ ตัวอย่างเช่น

```

1 def function_parameter():
2     print ("I am a parameter!")
3
4 function_dict = {"func": function_parameter}
5 func_para = function_dict['func']
6 func_para()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



I am a parameter!

OUTPUT

จากตัวอย่างโปรแกรมข้างบน บรรทัดที่ 1 โปรแกรมสร้างฟังก์ชันชื่อ function_parameter() ทำหน้าที่พิมพ์ข้อความว่า "I am a parameter!" จากนั้นบรรทัดที่ 4 โปรแกรมประกาศตัวแปรชื่อ function_dict ซึ่งเป็นตัวแปรชนิดดิกชันnaire ประกอบไปด้วยคีย์ชื่อ "func" และมีข้อมูลคือ function_parameter ซึ่งก็คือฟังก์ชัน function_parameter() นั่นเอง เมื่อโปรแกรมดึงข้อมูลจากตัวแปร function_dict ในบรรทัดที่ 5 โดยการกำหนดคีย์ "func" (function_dict['func']) ค่าที่ส่งคืนกลับมาจะเป็นอ็อปเจกต์ของฟังก์ชัน function_parameter() ส่งผลให้ตัวแปร func_para เป็นฟังก์ชัน function_parameter() ด้วย เมื่อเรียก func_para() ในบรรทัดที่ 6 จะทำให้ได้ผลลัพธ์เหมือนกับการเรียก function_parameter() เช่นเดียวกัน

การส่งฟังก์ชันเป็นพารามิเตอร์

เมื่อฟังก์ชันสามารถใช้เป็นข้อมูลหรือพารามิเตอร์ให้กับตัวแปรชนิดต่างๆ ได้ เช่น ดิกชันรายลิสต์ ทัพเพิล จะนั่นฟังก์ชันก็สามารถกำหนดเป็นพารามิเตอร์ให้กับฟังก์ชันอื่นๆ ได้ เช่นเดียวกัน เรียกฟังก์ชันที่รับฟังก์ชันอื่นๆ เป็นพารามิเตอร์ หรือคืนค่ากลับเป็นฟังก์ชันว่า "higher order function" ตัวอย่างเช่น

```

1 def self_absorbed_function():
2     return ("I'm an amazing function!")
3
4 def printer(func):
5     print ("The function passed to me says: " + func())
6
7 printer(self_absorbed_function)

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



The function passed to me says: I'm an amazing function!

จากตัวอย่างโปรแกรมข้างบน บรรทัดที่ 1 โปรแกรมสร้างฟังก์ชันชื่อ `self_absorbed_function()` ทำหน้าที่คืนค่าเป็นข้อความว่า "I'm an amazing function!" ไปยังผู้เรียกใช้งาน บรรทัดที่ 4 โปรแกรม สร้างฟังก์ชันชื่อ `printer()` ทำหน้าที่พิมพ์ข้อความว่า "The function passed to me says: " ตามด้วย ข้อความที่เกิดจากฟังก์ชันชื่อ `func()` นั่นก็คือผลลัพธ์ที่ได้จากฟังก์ชัน `self_absorbed_function()` นั่นเอง โดยฟังก์ชัน `printer()` ต้องการพารามิเตอร์ 1 ตัวซึ่งเป็นฟังก์ชัน เมื่อผู้ใช้งานเรียกฟังก์ชัน `printer(self_absorbed_function)` พร้อมกับส่งฟังก์ชันเป็นพารามิเตอร์ให้กับฟังก์ชันดังกล่าว ทำให้ได้ผลลัพธ์ดังแสดงในตัวอย่าง OUTPUT ด้านบน

Decorator คือ wrapper (ผู้ห่อหุ้ม)

จากที่กล่าวมาแล้วว่าฟังก์ชันสามารถเป็นพารามิเตอร์ให้กับฟังก์ชันอื่นๆ ได้ ซึ่งหลักการทำงาน นี้ถูกนำไปใช้กับ Decorator ด้วย คือ เมื่อส่งฟังก์ชันใดๆ ให้กับ Decorator ฟังก์ชันเหล่านั้นจะถูกแก้ไข ตัดต่อ หรือห่อหุ้มจากฟังก์ชัน Decorator นั่นเอง ตัวอย่างเช่น

```

⇒1 def logging_decorator(func):
⇒2     def wrapper():
3         wrapper.count += 1
4         print("The function I modify has been called {}"
5               .format(wrapper.count))
5         func()
6         wrapper.count = 0
7     return wrapper
8
⇒9 def a_function():
10    print("I'm a normal function.")
11
⇒12 modified_function = logging_decorator(a_function)
13 modified_function()
14 modified_function()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



The function I modify has been called 1 times(s).
I'm a normal function.
The function I modify has been called 2 times(s).
I'm a normal function.

จากตัวอย่างโปรแกรมข้างบน บรรทัดที่ 1 สร้างฟังก์ชัน `Decorator` ชื่อว่า `logging_decorator()` ทำหน้าที่ห่อหุ้มฟังก์ชันชื่อว่า `func()` โดยมีพารามิเตอร์เป็นฟังก์ชันชื่อ `func() = a_function()` นั่นเอง บรรทัดที่ 2 โปรแกรมสร้างฟังก์ชันชื่อ `wrapper()` ทำหน้าที่สร้างข้อความเพื่อครอบก่อนที่ฟังก์ชัน `func()`

จะทำงาน โดยพังก์ชันดังกล่าวทำหน้าที่เพิ่มค่าให้กับตัวแปร wrapper.count ขึ้นครั้งละ 1 พร้อมกับพิมพ์ข้อความว่า "The function I modify has been called {0} times(s)." โดย {0} หมายถึงข้อมูลที่ถูกเมะอัด format() จัดรูปแบบให้ โดย {0} = wrapper.count นั่นเอง บรรทัดที่ 5 โปรแกรมเรียก func() มาทำงาน ซึ่งหมายถึงเรียกพังก์ชัน a_function() มาทำงานนั่นเอง ในบรรทัดที่ 6 สร้างตัวแปรชื่อ wrapper.count เพื่อใช้ทำงานกับพังก์ชัน wrapper() เท่านั้น โดยใช้คำนำหน้าเป็นชื่อของพังก์ชัน (wrapper) และตามด้วยชื่อตัวแปร เช่น wrapper.count เป็นต้น บรรทัดที่ 7 โปรแกรมคืนค่ากลับเป็นพังก์ชัน wrapper() ให้กับผู้เรียกใช้งาน

บรรทัดที่ 9 สร้างพังก์ชันแบบธรรมดาก็ว่า a_function() ทำหน้าที่พิมพ์ข้อความว่า "I'm a normal function." บรรทัดที่ 12 เรียกใช้พังก์ชัน logging_decorator() โดยส่งพังก์ชัน a_function() เป็นพารามิเตอร์ให้กับพังก์ชันดังกล่าว ส่งผลให้ตัวแปร modified_function = พังก์ชัน wrapper() เมื่อเรียกใช้งานตัวแปร modified_function ตามด้วยเครื่องหมายวงเล็บ (...) ต่อท้ายตัวแปรดังกล่าว จะส่งผลให้เหมือนกับเป็นการเรียกใช้งานพังก์ชัน wrapper() นั่นเอง ผลลัพธ์ที่ได้แสดงในตัวอย่างข้างบน

สรุปคือ โปรแกรมจะพิมพ์ข้อความ "The function I modify has been called 1 times(s)." ก่อนแล้ว จึงพิมพ์ข้อความ "I'm a normal function." ตามมา เมื่อเรียกใช้ตัวแปร (ชนิดพังก์ชัน) modified_function อีกครั้งค่า wrapper.count จะเพิ่มขึ้นไปเรื่อยๆ จนกว่าจะยุติการทำงานของโปรแกรม

Decorator syntax (@)

การเรียกใช้งาน Decorator ในหัวข้อก่อนหน้า ทำได้โดยการประกาศชื่อตัวแปรเพื่อรับค่าอีกเจ้าต์จาก Decorator เช่น modified_function = logging_decorator(a_function) จากนั้นใช้ตัวแปรดังกล่าวเรียกใช้งานพังก์ชัน wrapper (modified_function()) ต่อไป แต่ผู้เขียนโปรแกรมสามารถทำให้โปรแกรมกระชับขึ้นได้ โดยใช้สัญลักษณ์ @ นำหน้าชื่อพังก์ชันที่ต้องการส่งเป็นพารามิเตอร์ให้กับ Decorator และเรียกใช้พังก์ชันแบบปกติโดยไม่ต้องอ้างถึงพังก์ชัน Docorator อีก เช่น

```

1  def logging_decorator(func):
2      count = 0
3      def wrapper():
4          wrapper.count += 1
5          print("The function I modify has been called {0}
times(s) .".format(wrapper.count))
6          func()
7      wrapper.count = 0
8      return wrapper
9
⇒10 @logging_decorator
⇒11 def a_function():
12     print("I'm a function.")

```

```

13
⇒14 @logging_decorator
⇒15 def some_function():
16     print("I'm some_function.")
17
⇒18 a_function()
⇒19 some_function()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



The function I modify has been called 1 times(s).
I'm a_function.
The function I modify has been called 1 times(s).
I'm some_function.

จากตัวอย่างด้านบน บรรทัดที่ 10 โปรแกรมประกาศว่า @logging_decorator คือ การอ้างถึงฟังก์ชัน Decorator ซึ่งว่า logging_decorator และจะใช้ฟังก์ชันชื่อ a_function() ในบรรทัดที่ 11 เป็น parameter ให้กับ logging_decorator ทำงาน และบรรทัดที่ 14 ก็ทำงานในลักษณะเดียวกัน แต่เปลี่ยนฟังก์ชันชื่อ some_function() เป็น parameter ให้กับ logging_decorator ทำงานแทน สำหรับการเรียกใช้งาน โปรแกรมไม่จำเป็นต้องสร้างตัวแปรใหม่เหมือนในตัวอย่างที่ผ่านมา แต่ให้เรียกชื่อฟังก์ชันแบบปกติ เช่นในบรรทัดที่ 18 และ 19 พอจะจะรู้โดยอัตโนมัติว่า เมื่อเรียกฟังก์ชัน a_function() และ some_function() และจะต้องส่งฟังก์ชันทั้งสองเป็น parameter ให้กับฟังก์ชัน Decorator ซึ่งว่า logging_decorator ทำงานทันที

8. เมธอด .format()

.format() เป็นเมธอดที่ใช้สำหรับจัดรูปแบบการแสดงผลข้อมูล เช่น สตริง (String) และจำนวนจริง (Floating-point) ซึ่งมีรูปแบบการทำงานดังนี้

การจัดรูปแบบของข้อความสตริง

`template.format(p0, p1, ... pN, k0=v0, k1=v1, ...)`

โดย template คือข้อความสตริงที่ต้องการแสดงผล, p0 – pN คือตำแหน่งที่ต้องการวางสตริงใน template ซึ่งโดยปกติสามารถกำหนดได้ 2 แบบคือ กำหนดโดยใช้เครื่องหมาย [...] เช่น {0}, {1}, ..., {N} หรือกำหนดโดยใช้ Keyword เช่น {city}, {name} และ k0=v0, k1=v1, kn=vn คือการกำหนดค่าให้กับ Keyword เช่น city= 'thailand', name = 'John' สำหรับการใช้งานแสดงได้ดังนี้

ตัวอย่างที่ 1:

```

>>> "We have {0} hectares planted to {1}.".format(49, "Apple")
'We have 49 hectares planted to Apple.'

```

```
"We have {0} hectares planted to {1}.".format(49, "Apple")
```

ตัวอย่างที่ 2:

```
>>> "{monster} has now eaten {city}.".format(city='Tokyo',
monster='Godzillar')
Godzillar has now eaten Tokyo
```

```
"{monster} has now eaten {city}.".format(city='Tokyo', monster='Godzillar')
```

ตัวอย่างที่ 3:

```
>>> "The {structure} sank {0} times in {1} years.".format(3,
2, structure='castle')
The castle sank 3 times in 2 years.
```

การจัดรูปแบบของจำนวนจริง

{x:.yf}.format(floating-point)

โดย {...} ใช้ระบุขอบเขตของจำนวนจริง, x แสดงจำนวนจริงหน้าจุดทศนิยม, : คือจุดเริ่มต้นของจุดทศนิยม, y คือจำนวนเลขทศนิยมหลังจุดทศนิยม และ f คือเลขจำนวนจริง สำหรับการใช้งานแสดงได้ดังนี้

ตัวอย่างที่ 1:

```
>>> "{0:.4f}".format(0.1234567890)
'0.1235'
>>> "{0:.4f}".format(10.1234567890)
'10.1235'
```

ตัวอย่างที่ 2:

```
>>> "{0:8.4f}".format(0.1234567890) #กำหนดเลขหน้าจุดทศนิยมเท่ากับ 8 ตัว
' 0.1235'
>>> "{0:8.4f}".format(10.987654321)
' 10.9877'
```

ตัวอย่างที่ 3:

```
>>> "{0:.4e}".format(0.1234567890) #กำหนดเป็น exponential  
'1.2346e-01'
```

ตัวอย่างที่ 4:

```
>>> import math  
>>> "sin({0:.4f}) = {1:.4e}".format(0.1234567890,  
math.sin(0.123456789))  
'sin(0.1235) = 1.2314e-01'
```

9. อาเรย์และเมทริกซ์ (Array and Matrix)

ไฟรอนมีพันธมิตรที่ช่วยกันพัฒนาความสามารถต่างๆ ให้กับไฟรอนมากมาย ซึ่งรวมถึง ความสามารถในการประมวลผลคณิตศาสตร์ชั้นสูงด้วย เช่น อาเรย์หลายมิติ, เมทริกซ์ขนาดใหญ่, Optimization, Regression และ Fourier-transformation (มีความสามารถคล้าย MATLAB) เป็นต้น ซึ่ง โมดูลเหล่านี้คือ SciPy (Scientific Python) และ NumPy โดยปกติโมดูลทั้งสองไม่ได้ติดตั้งมาพร้อมกับไฟรอน จำเป็นต้องติดตั้งเสียก่อน สามารถดาวน์โหลดได้จาก URL: <http://www.scipy.org/> หรือจาก CD เมื่อติดตั้ง SciPy และ NumPy เสร็จเรียบร้อยแล้ว ทดสอบการใช้งานดังนี้

อาเรย์กับ NumPy โมดูล

- สร้างอาเรย์ชนิดจำนวนเต็ม 1 มิติที่มีขนาด 1 ถึง 4 คอลัมว

```
>>> import numpy as np  
>>> x = np.array([42, 47, 11, -55], int)  
>>> x  
array([42, 47, 11, -55])  
>>> print(x[2]) #การเข้าถึงอาเรย์แถวที่ 0 คอลัมที่ 2  
11
```

- สร้างอาเรย์ชนิดจำนวนเต็ม 2 มิติ ขนาด 3 ถึง 3 คอลัมว

```
>>> x = np.array(((11, 12, 13), (21, 22, 23), (31, 32, 33)))  
>>> print(x)  
[[11 12 13]  
 [21 22 23]  
 [31 32 33]]  
>>> print(x[1][1]) #การเข้าถึงอาเรย์แถวที่ 1 คอลัมที่ 1  
22
```

- สร้างอาเรย์ชนิดจำนวนเต็ม 3 มิติ ขนาด 2 ถึง 2 คอลัมว และลีก 2 หน่วย

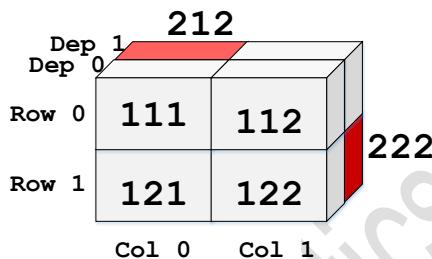
```

>>> x = np.array(((111,112), (121,122)),
   ((211,212), (221,222)))
>>> print (x)
[[[111 112]
 [121 122]

 [211 212]
 [221 222]]]

>>> print (x[1][0][1]) #การเข้าถึงอาร์เรย์แถวที่ 1 คอลัมที่ 0 และความลึก 1 หน่วย
212
>>> print (x[1][1][1]) #การเข้าถึงอาร์เรย์แถวที่ 1 คอลัมที่ 1 และความลึก 1 หน่วย
222

```



Note: อาร์เรย์ในทางทฤษฎีจะอ้างถึงข้อมูลเริ่มจาก แถวที่ 1, คอลัมที่ 1 แต่ในทางคอมพิวเตอร์จะอ้าง แถวที่ 0, คอลัมที่ 0 แทน ในหนังสือเล่มนี้จะอ้างตำแหน่งตามปกติ ข้อมูลในอาร์เรย์แบบคอมพิวเตอร์ จะนับเริ่มต้นจาก 0

- แสดงจำนวนมิติของอาร์เรย์ โดยใช้แอ็ตทริบิวต์ ndim

```

>>> x = np.array( ((11,12,13), (21,22,23) ))
>>> x.ndim
2

```

- แสดงจำนวนสมาชิกของอาร์เรย์ โดยใช้เมธอด shape

```

>>> x = np.array(((7,3,0), (7,11,9), (15,3,7),(12,4,8)))
>>> x.shape
(4, 3)

```

- แสดงช่วงสมาชิกของอาร์เรย์โดยใช้สัญลักษณ์ [X:Y] โดย X คือแถว และ Y คือคอลัม

```

>>> x = np.array([42,47,11])
>>> x[:2]
array([42, 47])      #แสดงสมาชิกตั้งแต่คอลัมที่ 0 - 1
>>> x[0]            #แสดงข้อมูลสมาชิกตำแหน่งคอลัมที่ 0
42
>>> x[1] = 49 #กำหนดค่าของข้อมูลสมาชิกในตำแหน่งคอลัมที่ 1

```

```
>>> x
array([42, 49, 11])
```

- สร้างอาร์เรย์ชนิดจำนวนวันจริง 2 มิติ ขนาด 2 และ 2 คอลัมก์

```
>>> x = np.array([[0.314, 0.5, 17], [1.4142, 0.67, 7]], float)
>>> x
array([[ 0.314,   0.5,   17.],
       [ 1.4142,  0.67,    7.]])
>>> x[0] #แสดงข้อมูลสมาชิกแถวที่ 0
array([ 0.314,   0.5,   17.])
>>> x[1] #แสดงข้อมูลสมาชิกแถวที่ 1
array([ 1.4142,  0.67,    7.])
>>> x[0,2]      #แสดงข้อมูลสมาชิกแถวที่ 0 คอลัมก์ที่ 2
17.0
>>> x[:,0]      #แสดงข้อมูลสมาชิกทุกๆ แถว คอลัมก์ที่ 0
array([ 0.314,  1.4142])
>>> x[:,1]      #แสดงข้อมูลสมาชิกทุกๆ แถว คอลัมก์ที่ 1
array([ 0.5,  0.67])
>>> x[:,2]      #แสดงข้อมูลสมาชิกทุกๆ แถว คอลัมก์ที่ 2
array([ 17.,    7.])
```

- การเปลี่ยนรูปร่างของอาร์เรย์ด้วยเมธอด flatten() และ ravel()

เมธอด flatten() และ ravel() ทำหน้าที่รับเปลี่ยนรูปร่างของอาร์เรย์ให้อยู่ในแนวราบ

```
>>> import numpy as np
>>> x = np.array([[ [ 0,  1],
                  [ 2,  3],
                  [ 4,  5],
                  [ 6,  7]],
                 [[ 8,  9],
                  [10, 11],
                  [12, 13],
                  [14, 15]],
                 [[16, 17],
                  [18, 19],
                  [20, 21],
                  [22, 23]]])
>>> x.flatten()
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
       13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
```

- การเปลี่ยนรูปร่างของอาร์เรย์ให้มีมิติตามที่ต้องการด้วยเมธอด reshape() โดยอาร์เรย์ต้นฉบับจะไม่มีการเปลี่ยนแปลง

```
>>> x = np.array(range(24))      #สร้างอาร์เรย์ 1 มิติที่มีสมาชิกตั้งแต่ 0 - 23
>>> x
```

```

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
       13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
>>> y = x.reshape((3,4,2)) #เปลี่ยนรูปทรงอาร์เรย์เป็น 3 มิติ 4 แถว 2 คอลัมว์ ลึก 3
>>> y
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5],
        [ 6,  7]],

       [[ 8,  9],
        [10, 11],
        [12, 13],
        [14, 15]],

       [[[16, 17],
        [18, 19],
        [20, 21],
        [22, 23]]])

>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
       13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]) #อาร์เรย์ไม่เปลี่ยนรูปทรง

```



Note: เมธอด reshape() จะปรับขนาดของอาร์เรย์ โดยมีรูปแบบคือ reshape(deep, row, col) โดย deep = ขนาดความลึกของอาร์เรย์, row = จำนวนแถว, col = จำนวนคอลัมว์

- การเชื่อมอาร์เรย์เข้าด้วยกันโดยใช้เมธอด concatenate()

```

>>> x = np.array([11,22])
>>> y = np.array([18,7,6])
>>> z = np.array([1,3,5])
>>> np.concatenate((x,y,z))
array([11, 22, 18,  7,  6,  1,  3,  5])

```

- เพิ่มมิติใหม่ให้กับอาร์เรย์ โดยใช้แอตทริบิวต์ newaxis

```

>>> x = np.array([2,5,18,14,4])
>>> x
array([ 2,  5, 18, 14,  4])
>>> y = x[:, np.newaxis]
>>> y
array([[ 2],
       [ 5],
       [18],
       [14],
       [ 4]])

```

- กำหนดค่าเริ่มต้นเป็น 0 และ 1 ให้กับอาร์เรย์

```

>>> np.ones((2,3)) #กำหนดค่าเริ่มต้นให้กับอาร์เรย์ 2 x 3 เป็น 1.0 (float)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])

```

```
>>> np.ones((3, 4), dtype=int) #กำหนดค่าเริ่มต้นให้กับอาร์ย์ 2 x 3 เป็น 1 (int)
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
>>> np.zeros((2, 4)) #กำหนดค่าเริ่มต้นให้กับอาร์ย์ 2 x 4 เป็น 0.0 (float)
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

- รีเซ็ต (Reset) ข้อมูลสมาชิกในอาร์ย์ให้เป็น 1 หรือ 0 ด้วยเมธอด `ones_like()` และ `zeros_like()`

```
>>> x = np.array([2, 5, 18, 14, 4])
>>> np.ones_like(x)
array([1, 1, 1, 1, 1])
>>> np.zeros_like(x)
array([0, 0, 0, 0, 0])
```

เลขคณิตเมตริกซ์

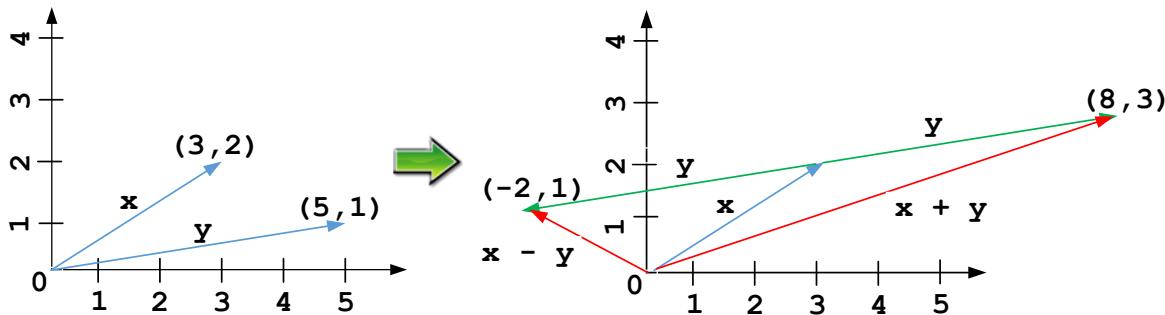
ในหัวข้อที่ผ่านมาได้กล่าวถึงการสร้างอาร์ย์ในหลากหลายรูปแบบ ซึ่งการสร้างอาร์ย์เป็นพื้นฐานของการสร้างเมตริกซ์ด้วย ในบทนี้จะกล่าวถึงการประมวลผลคณิตศาสตร์กับเมตริกซ์ เช่น การบวกเมตริกซ์, การลบ, การคูณ, Skalar product, Cross product และการดำเนินการอื่นๆ ไฟชอนใช้ตัวดำเนินการพื้นฐานคือ + (บวก), - (ลบ), * (คูณ), / (หาร), ** (ยกกำลัง), % (modulo) กับอาร์ย์

- การดำเนินการพื้นฐานใดๆ ขนาดของอาร์ย์จะต้องมีขนาดเท่ากัน

```
>>> x = np.array([1, 5, 2])
>>> y = np.array([7, 4, 1])
>>> x + y
array([8, 9, 3])
>>> x * y
array([ 7, 20,  2])
>>> x - y
array([-6,  1,  1])
>>> x / y
array([0, 1, 2])
>>> x % y
array([1, 1, 0])
```

- การรวมเวกเตอร์

การรวมเวกเตอร์ หมายถึง การบวกหรือลบกันของเวกเตอร์ตั้งแต่ 2 เวกเตอร์ ขึ้นไป ผลลัพธ์ที่ได้เป็นปริมาณเวกเตอร์ เรียกว่า เวกเตอร์ลัพธ์ (Resultant Vector) ซึ่งพิจารณาได้ ดังนี้



```
>>> x = np.array([3,2])
>>> y = np.array([5,1])
>>> z = x + y
>>> z
array([8, 3])
>>> z = x - y
>>> z
array([-2, 1])
```

- การคูณสเกลาร์ (Scalar product หรือ dot product แทนด้วยเครื่องหมาย ". ")

จากนิยามของการคูณสเกลาร์ $\vec{A} \cdot \vec{B} = |A||B| \cos\theta(\vec{A}, \vec{B}) = \vec{A} \cdot \vec{B} = A_1B_1 + A_2B_2 + A_3B_3$

```
>>> x = np.array([1,2,3])
>>> y = np.array([-7,8,9])
>>> np.dot(x,y)
36
>>> dot = np.dot(x,y)
>>> x_modulus = np.sqrt((x*x).sum())
>>> y_modulus = np.sqrt((y*y).sum())
>>> cos_angle = dot / x_modulus / y_modulus #cosinus of
angle between x and y
>>> angle = np.arccos(cos_angle)
>>> angle
0.80823378901082499
>>> angle * 360 / 2 / np.pi # angle in degrees
46.308384970187326
```

- การคูณเมทริกซ์สามารถทำได้โดยการใช้เมธอด matrix() ร่วมกับเครื่องหมาย *

```
>>> x = np.matrix( ((2,3), (3, 5)) )
>>> y = np.matrix( ((1,2), (5, -1)) )
>>> x * y
matrix([[17,  1],
       [28,  1]])
```



Note: ไม่สามารถใช้อาร์เรย์ทำการคูณเมทริกซ์ได้โดยตรง เช่น

```
>>> x = np.array( ((2,3), (3, 5)) )
>>> y = np.array( ((1,2), (5, -1)) )
>>> x * y
array([[ 2,  6],
       [15, -5]])
```

10. การสร้างเน็ตเวิร์คกราฟด้วย NetworkX

NetworkX เป็นโมดูลที่เสริมความสามารถของ Python ที่เกี่ยวข้องกับการบริหารจัดการเน็ตเวิร์ค กราฟ เพื่อใช้สำหรับวิเคราะห์ข้อมูลต่างๆ ที่จำเป็นต้องอาศัยความสามารถของกราฟช่วยในการทำงาน NetworkX ไม่ได้ติดตั้งมาพร้อมกับ Python ดังนั้นผู้เขียนจำเป็นต้องติดตั้งโปรแกรม NetworkX เพิ่มเติม โดยดาวน์โหลดได้จาก URL: <http://networkx.github.io/> หรือจาก CD ซึ่งเวอร์ชันล่าสุดคือ networkx-0.33.win32 ให้ทำการติดตั้งโดยคลิก Next > ไปเรื่อยๆ จนกว่าจะเสร็จสิ้นการติดตั้ง เมื่อติดตั้งเสร็จจะสามารถเรียกใช้จาก Python ได้ทันที

- เริ่มต้นการสร้างกราฟ

ในอันดับแรกจะแนะนำการสร้าง empty graph ซึ่งมีรูปแบบคำสั่งดังนี้ (NetworkX_1.py)

```

⇒1 import networkx as nx
2
⇒3 G=nx.Graph()
4
⇒5 print(G.nodes())
⇒6 print(G.edges())
7
⇒8 print(type(G.nodes()))
9  print(type(G.edges()))

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



OUTPUT

```

[ ]
[ ]
<type 'list'>
<type 'list'>

```

จากตัวอย่างด้านบน บรรทัดที่ 1 นำเข้าโมดูล networkx สำหรับใช้สร้างกราฟ โดยเปลี่ยนเป็นชื่อเล่นคือ nx บรรทัดที่ 3 สร้างอ็อปเจกต์ของกราฟชื่อว่า G ซึ่งเป็นกราฟว่าง บรรทัดที่ 5, 6 ทดสอบพิมพ์ค่าของ โหนดและเส้นกราฟ ผลลัพธ์ที่ได้คือ กราฟที่ไม่มีโหนดและเส้นเชื่อมใดๆ ข้อมูลที่ใช้เก็บโหนดและเส้น เชื่อมจะเป็นข้อมูลชนิดลิสต์

- การเพิ่มโหนดให้กราฟ

การเพิ่มโหนดให้กับกราฟจะใช้เมธอด add_node(node_name) และเพิ่มโหนดพร้อมๆ กัน หลายๆ โหนดจะใช้คำสั่ง add_nodes_from(list_node) เช่น (NetworkX_2.py)

```

1 import networkx as nx
2
3 G=nx.Graph()
4
5 # adding just one node:
⇒6 G.add_node("a")

```

```

7   # a list of nodes:
8   G.add_nodes_from(["b", "c"])
9
10  print("Nodes of graph: ")
11  print(G.nodes())
12  print("Edges of graph: ")
13  print(G.edges())

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



Nodes of graph:

['a', 'c', 'b']

Edges of graph:

[]

จากตัวอย่างด้านบน บรรทัดที่ 6 แสดงคำสั่งสำหรับเพิ่มโหนดให้กับกราฟเพียง 1 โหนดเท่านั้น และบรรทัดที่ 7 แสดงการเพิ่มโหนดหลายๆ โหนดพร้อมๆ กัน

- การเพิ่มเส้นเชื่อม (edges) ให้กับกราฟ

การเพิ่มเส้นเชื่อมของกราฟสามารถทำได้โดยใช้เมธอด add_edge(startnode, endnode) สำหรับเพิ่ม 1 เส้น แต่ถ้าต้องการเพิ่มเส้นเชื่อมมากกว่า 1 เส้นให้ใช้ * แทน เช่น (NetworkX_3.py)

```

1 import networkx as nx
2
3 G=nx.Graph()
4 G.add_node("a")
5 G.add_nodes_from(["b", "c"])
6
7 G.add_edge(1, 2)
8 e = ("d", "e")
9 G.add_edge(*e)
10 e = ("a", "b")
11 G.add_edge(*e)
12
13 print("Nodes of graph: ")
14 print(G.nodes())
15 print("Edges of graph: ")
16 print(G.edges())

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



Nodes of graph:

['a', 1, 'c', 'b', 'e', 'd', 2]

Edges of graph:

[('a', 'b'), (1, 2), ('e', 'd')]

จากตัวอย่างด้านบน บรรทัดที่ 7 แสดงการเพิ่มเส้นเชื่อมลงในกราฟโดยใช้คำสั่ง G.add_edge(1, 2) (สร้างเส้นเชื่อมระหว่างโหนดที่ 1 และ 2) แต่ในตัวอย่างนี้ยังไม่เคยสร้างโหนด 1 และ 2 มา ก่อน ดังนั้น เมื่อใช้คำสั่งเพิ่มเส้นเชื่อม โปรแกรมจะสร้างโหนด 1 และ 2 ให้อัตโนมัติ บรรทัดที่ 8 โปรแกรมสร้างเส้น

เชื่อมจาก "d" ไป "e" ด้วยตัวแปรชนิดทัพเพิลชื่อ e จากนั้นนำตัวแปรดังกล่าวเป็นพารามิเตอร์ให้กับเมธอด add_edge(*e) ในบรรทัดที่ 9 ซึ่งสัญลักษณ์ * หมายถึงสามารถเพิ่มเส้นเชื่อมได้มากกว่า 1 เส้น บรรทัดที่ 10 และ 11 ก็เป็นวิธีการเพิ่มเส้นเชื่อมเดียวกัน แต่เชื่อมระหว่าง "a" ไป "b" แทน ดังนั้น ผลลัพธ์ที่ได้ในตัวอย่าง OUTPUT ข้างบนคือ โหนดในกราฟจะมีทั้งหมด 7 โหนด ('a', 1, 'c', 'b', 'e', 'd', 2) และมีเส้นเชื่อมทั้งหมด 3 เส้นคือ เส้นเชื่อมจาก 'a' ไปยัง 'b', 1 ไปยัง 2 และ 'e' ไปยัง 'd'

เมื่อต้องการเพิ่มเส้นเชื่อมมากกว่า 1 เส้นสามารถใช้เมธอด add_edges_from() ดังนี้

```
G.add_edges_from([('a','c'),('c','d'), ('a',1), (1,"d"), ("a",2)])
```

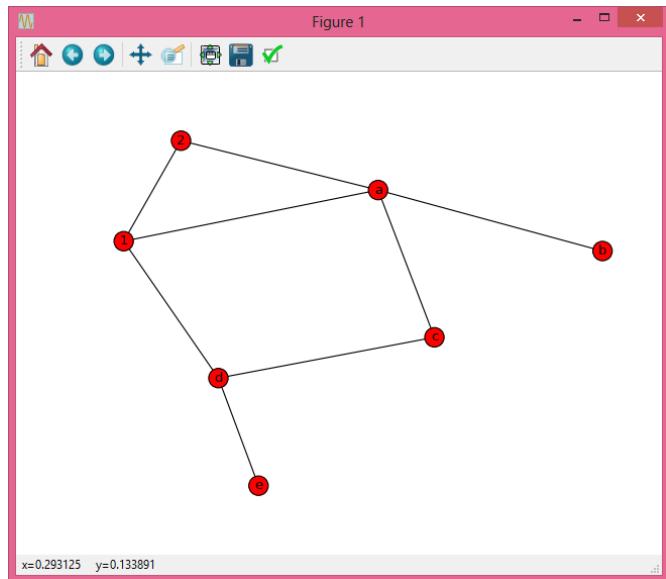
และสามารถ plot เป็นกราฟได้โดยใช้งานร่วมกับโมดูล matplotlib ดังนี้ (NetworkX_4.py)

```

1 import networkx as nx
2 import matplotlib.pyplot as plt
3
4 G=nx.Graph()
5 G.add_node("a")
6 G.add_nodes_from(["b", "c"])
7
8 G.add_edge(1,2)
9 e = ("d", "e")
10 G.add_edge(*e)
11 e = ("a", "b")
12 G.add_edge(*e)
13 G.add_edges_from([(("a", "c"), ("c", "d"), ("a", 1), (1,"d"),
14 ("a", 2))])
15 print("Nodes of graph: ")
16 print(G.nodes())
17 print("Edges of graph: ")
18 print(G.edges())
19
20 nx.draw(G)
21 plt.savefig("simple_path.png") # save as png
22 plt.show() # display

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



11. การพรีวิวต์กราฟ (Plot Graph) ด้วย MATPLOTLIB, NUMPY และ SCIPY

“ไฟรอน”ได้รับการสนับสนุนด้านการพัฒนากราฟโดยใช้โมดูล MATPLOTLIB, NUMPY และ SCIPY ทำงานร่วมกัน ซึ่งโมดูลเหล่านี้ไม่ได้ติดตั้งมาพร้อมกับไฟรอน จึงจำเป็นต้องติดตั้งเพิ่มทีหลัง แต่ในปัจจุบันได้มีการรวบรวมโมดูลเหล่านี้ไว้เป็นแพคเกจเดียวกันชื่อว่า Anaconda-1.9.1-Windows-x86_64.exe ขนาดประมาณ 370 MB หรือคันหาได้จาก CD หรือสามารถโหลดได้จาก URL:

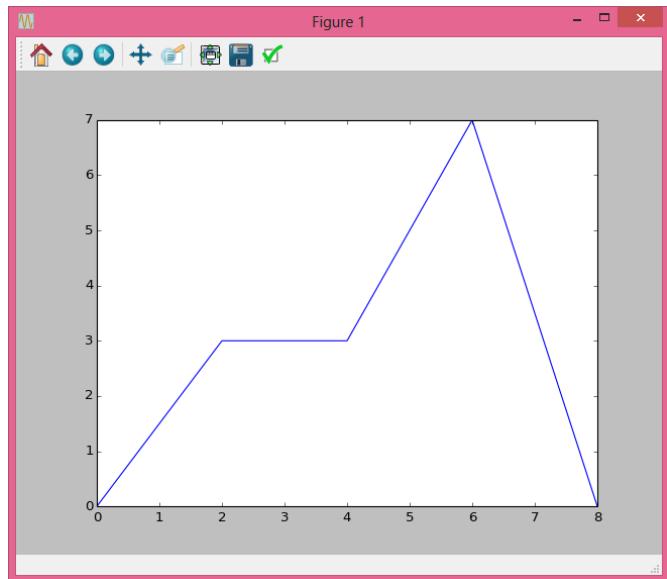
<http://continuum.io/downloads> เมื่อดาวน์โหลดเสร็จเรียบร้อยแล้ว ให้ทำการติดตั้งโปรแกรมซึ่งสามารถคลิก Next > ไปเรื่อยๆ จนกว่าจะเสร็จสิ้น เมื่อติดตั้งเสร็จสิ้นแล้วจะปรากฏโปรแกรมชื่อว่า anaconda command prompt ให้ผู้ใช้คลิกเพื่อเปิดโปรแกรม และจะปรากฏ prompt เพื่อรับคำสั่งจากผู้ใช้ซึ่งโปรแกรมดังกล่าวมีโปรแกรมทั้งหมดรวมอยู่ คือ Python, MATPLOTLIB, NUMPY และ SCIPY

- เริ่มต้นการพรีอตกราฟ

ให้เขียนโปรแกรมดังต่อไปนี้ แล้วบันทึกแฟ้มชื่อ plot1.py และเก็บไว้ในไดเรคทรอรี่ของ anaconda เช่น C:\Users\UserName\Anaconda และสั่งรันโปรแกรมดังนี้ python plot1.py

```
1 import matplotlib.pyplot as plt  
2 x = [0, 2, 4, 6, 8]  
3 y = [0, 3, 3, 7, 0]  
4 plt.plot(x, y)  
5 plt.show()  
6 plt.savefig("MyFirstPlot.png")
```

ผลลัพธ์ที่ได้มีความสั่งรันໂโปรแกรมคือ



เมื่อรันโปรแกรมแล้วกราฟที่ถูกสร้างขึ้นจะบันทึกเป็นไฟล์รูปภาพชื่อว่า "MyFirstPlot.png" ด้วยในไดเรกทรอรีปัจจุบัน

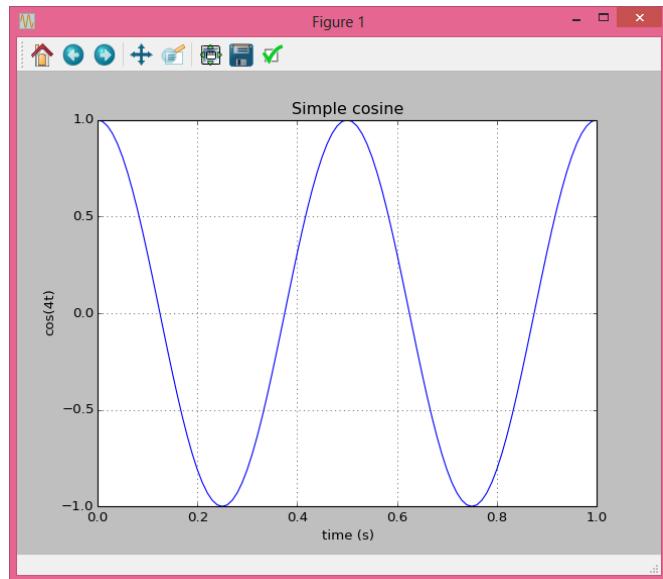
- แสดงการพร็อตกราฟชนิด Sine Wave (plot2.py)

```

1 import numpy
2 import pylab
3
4 t = numpy.arange(0.0, 1.0+0.01, 0.01)
5 s = numpy.cos(numpy.pi*4*t)
6 pylab.plot(t, s)
7
8 pylab.xlabel('time (s)')
9 pylab.ylabel('cos(4t)')
10 pylab.title('Simple cosine')
11 pylab.grid(True)
12 pylab.savefig('simple_cosine')
13
14 pylab.show()

```

- ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



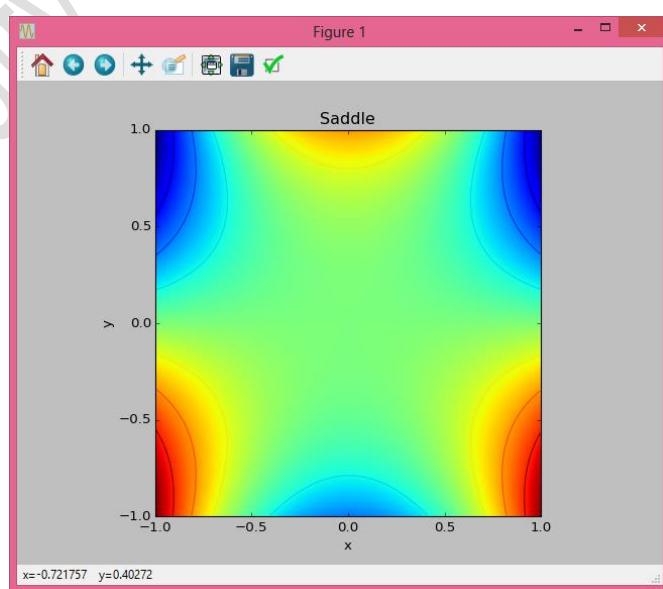
- แสดงการพรีอต Contour (plot3.py)

```

1 import scipy
2 import pylab
3 import matplotlib.pyplot as plt
4
5 x,y = scipy.ogrid[-1.:1.:01, -1.:1.:01]
6 z = x**3-3*x*y**2
7 pylab.imshow(z, origin='lower', extent=[-1,1,-1,1])
8 plt.contour(z, origin='lower', extent=[-1,1,-1,1])
9 pylab.xlabel('x')
10 pylab.ylabel('y')
11 pylab.title('Saddle')
12 pylab.savefig('Saddle')
13 plt.show()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



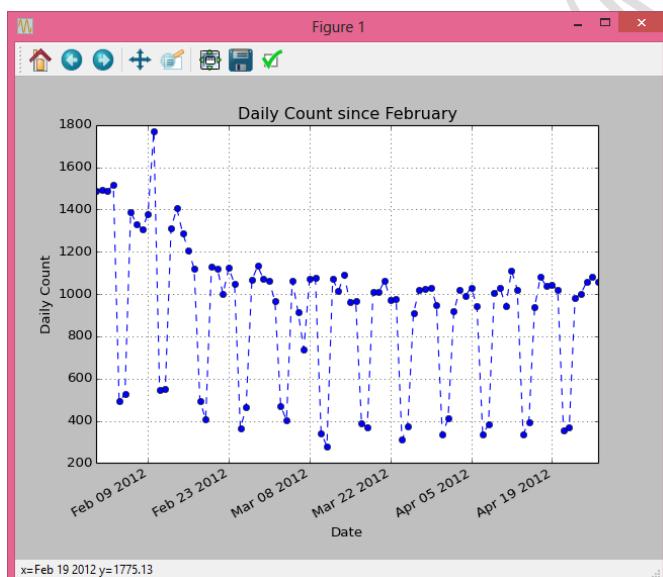
- แสดงการพร็อตโดยใช้ข้อมูลจากแฟ้ม .CSV (plot4.py)

```

1 import scipy
2 import pylab
3 import matplotlib.pyplot as plt
4
5 x,y = scipy.ogrid[-1.:1.:01, -1.:1.:01]
6 z = x**3-3*x*y**2
7 pylab.imshow(z, origin='lower', extent=[-1,1,-1,1])
8 plt.contour(z, origin='lower', extent=[-1,1,-1,1])
9 pylab.xlabel('x')
10 pylab.ylabel('y')
11 pylab.title('Saddle')
12 pylab.savefig('Saddle')
13 plt.show()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมคือ



ภาษา Pythonสนับสนุนให้สามารถพัฒนาโปรแกรมเพื่อแก้ปัญหาได้มากมาย ซึ่งผู้เขียนไม่สามารถกล่าวไว้ในหนังสือเล่มนี้ได้ทั้งหมด แต่ผู้เขียนหวังเป็นอย่างยิ่งว่าเนื้อหาทั้งหมดในหนังสือเล่มนี้จะเพียงพอต่อการทำความเข้าใจและเป็นพื้นฐานให้กับผู้อ่านสามารถนำไปต่อยอดได้ในอนาคต

จบภาค 3

Faculty of Informatics (MSU)



ภาคที่ 4

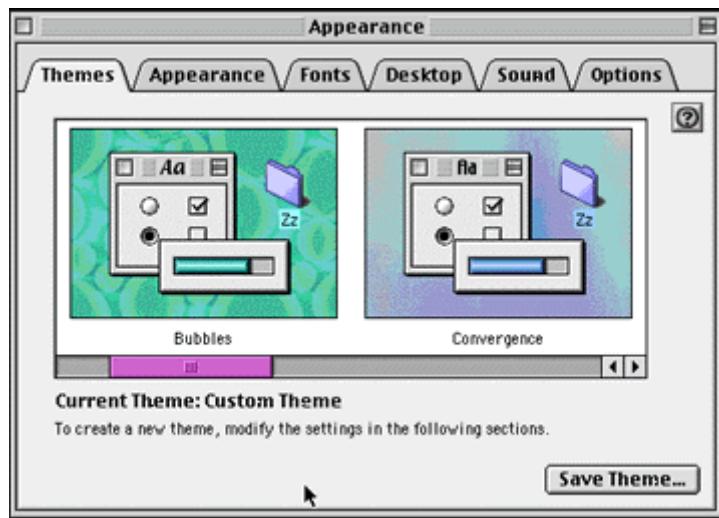
การเขียนโปรแกรมกราฟฟิกด้วยไพธอน



บทที่ 19 การเขียนโปรแกรมกราฟฟิกด้วยไพธอน

บทที่ 19

การเขียนโปรแกรมกราฟฟิกด้วยไพธอน (Graphics Programming with Python)



รูปแบบการปฏิสัมพันธ์กับระบบคอมพิวเตอร์ในปัจจุบันมี 2 รูปแบบใหญ่ๆ คือ แบบป้อนคำสั่งที่ละบรรทัด (Command line interface) หมายถึง การติดต่อสื่อสารระหว่างผู้ใช้กับระบบคอมพิวเตอร์ด้วย การป้อนคำสั่ง (Command) จากแป้นพิมพ์เพื่อให้ระบบปฏิบัติการประมวลผลตามที่ผู้ใช้ต้องการ เช่น MS-DOS และ ยูนิกซ์-ลินุกซ์ เป็นต้น และแบบกราฟฟิก คือ การติดต่อสื่อสารกับระบบคอมพิวเตอร์โดย อาศัยอินเตอร์เฟสแบบกราฟฟิก ซึ่งประกอบไปด้วยเมนู รูปภาพ ไอคอน โดยการใช้เมาส์คลิกเพื่อ สั่งงาน เช่น ระบบปฏิบัติการวินโดว์ และระบบปฏิบัติการ Machintosh เป็นต้น ในบทที่ผ่านๆ มาจะ เป็นการใช้งานด้วยวิธีการแบบป้อนคำสั่ง (Command line) เป็นส่วนใหญ่ แต่สำหรับในบทนี้จะอธิบาย การสร้างอินเตอร์เฟสแบบกราฟฟิกด้วยภาษาไพธอนเป็นหลัก

ไพธอนได้รับการสนับสนุนโมดูลสำหรับพัฒนา GUI (Graphic User Interface) ไว้มากมาย เช่น Tkinter, wxPython, JPython, wxWidgets, Qt, Gtk+, FLTK, FOX และ OpenGL เป็นต้น ซึ่งมี รายละเอียดเฉพาะโมดูลที่สำคัญๆ ดังนี้คือ

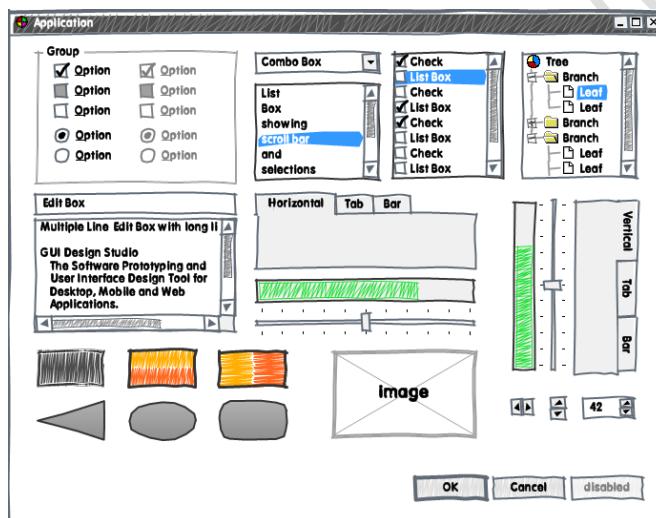
- Tkinter เป็นโมดูลที่พัฒนามาจาก Tk GUI Toolkit ซึ่งทำงานอยู่บนระบบปฏิบัติการยูนิกซ์มาก่อน ไพธอนได้เลือกโมดูลนี้ในการพัฒนากราฟฟิกบนไพธอนเป็นหลัก
- wxPython เป็นโมดูลที่ถูกพัฒนาขึ้นเพื่อให้ทำงานอยู่บนระบบปฏิบัติการวินโดว์เป็นหลัก และ เป็นลิขสิทธิ์แบบ Open source อ่านข้อมูลเพิ่มเติมได้ที่ <http://wxpython.org>

- JPython เป็นโมดูลที่พัฒนาขึ้นเพื่อให้ Python สามารถทำงานร่วมกับกราฟฟิกของภาษา Java ได้อ่านข้อมูลเพิ่มเติมได้ที่ <http://www.jython.org>
- อื่นๆ สามารถอ่านได้จาก <https://wiki.python.org/moin/GuiProgramming>

สำหรับในหนังสือเล่มนี้จะเน้นการสร้าง GUI ด้วย Tkinter เป็นหลัก

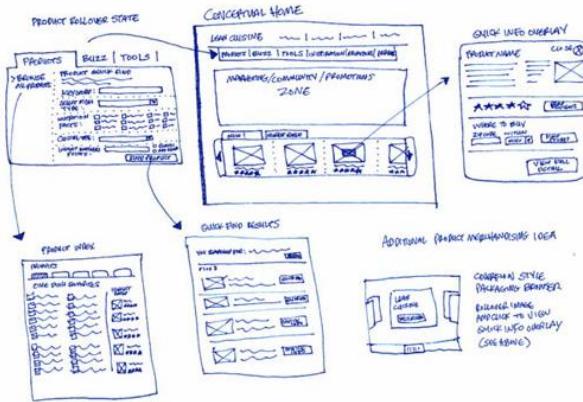
1. แนวความคิดในการออกแบบ GUI

ก่อนการพัฒนา GUI ด้วย Python ผู้เขียนโปรแกรมจำเป็นต้องทำการออกแบบหน้าตาของ GUI เสียก่อน โดยการสเก็ตช์ (Sketch) คือ การออกแบบโปรแกรมเป็นภาพบนกระดาษ หรือออกแบบด้วยโปรแกรมสำหรับสร้างงานกราฟฟิกด้วยคอมพิวเตอร์อย่างคร่าวๆ เสียก่อน ดังรูปที่ 19.1



รูปที่ 19.1 การสเก็ตช์ GUI อย่างคร่าวๆ บนกระดาษ

จากรูปที่ 19.1 แสดงการออกแบบ GUI บนกระดาษแบบหยาบๆ เพื่อให้เห็นภาพรวมของโปรแกรมและหน้าตาคร่าวๆ ก่อนการเขียนโปรแกรม ภาพที่สเก็ตช์ควรจะมีรายละเอียดต่างๆ เช่น หน้าต่างหลักของโปรแกรม (Main window) เมนูต่างๆ ที่สอดคล้องกับงาน ปุ่ม แท็บ ไอคอน เมนูย่อย และภาพ เป็นต้น จากนั้นทำการเชื่อมโยงหน้าต่างหรือเมนูต่างๆ เข้าไว้ด้วยกัน โดยใช้เส้นและลูกศร ดังรูปที่ 19.2

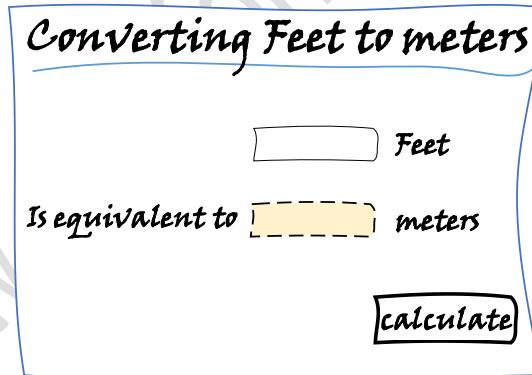


รูปที่ 19.2 การเชื่อมโยงหน้าต่างหรือเมนูต่างๆ ที่สัมพันธ์กัน

สำหรับตัวอย่างการเชื่อมโยงระหว่างหน้าต่างเข้าด้วยกัน เช่น เมื่อผู้ใช้งานกดปุ่ม Save as... โปรแกรมจะประมวลผลหน้าต่างเพื่อให้ผู้ใช้เลือกตำแหน่งได้โดยทอรี่ที่ต้องการบันทึกแฟ้มลงบน ชาร์ดดิสก์เป็นต้น

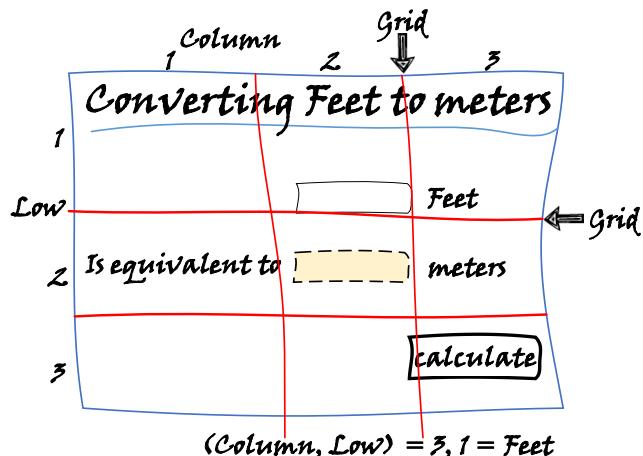
ตัวอย่างการออกแบบ

สมมุติว่าผู้เขียนโปรแกรมต้องการออกแบบ GUI อย่างง่าย เพื่อทำหน้าที่แปลงค่าหน่วยวัด ระยะทางจากฟุต (feet) เป็นเมตร (meters) ตัวอย่างรูปที่สเก็ตช์ได้แสดงดังรูปที่ 19.3



รูปที่ 19.3 แสดงภาพสเก็ตช์ของโปรแกรมแปลงจาก feet เป็น meters

จากรูปที่ 19.3 ผู้ใช้จะป้อนตัวเลขจำนวนจริงในช่อง Feet เมื่อกดปุ่ม calculate โปรแกรมจะทำการแปลงค่าจำนวนจริงที่ป้อนให้กับโปรแกรมเป็นเมตรและแสดงใน meters ขั้นตอนต่อไป ให้ผู้เขียนโปรแกรมทำการวัดเส้นกริดเพื่อใช้กำหนดขอบเขต สำหรับจัดวางตำแหน่ง (Layout) ของอوبเจกต์ต่างๆ ในโปรแกรม จากตัวอย่างด้านบน จะทำการกำหนดเส้นกริดเป็น 3 แก้ 3 คอลัมน์ดังรูปที่ 19.4



รูปที่ 19.4 แสดงการวาดเส้นกริด 3×3 ลงบนภาพสเก็ตช์

สำหรับตัวอย่างโปรแกรมแปลงฟุตเป็นเมตร แสดงในตัวอย่างที่ 19.1

Program Example 19.1: Converting Feet to Meters

```

⇒1 from tkinter import *
2
3
⇒4 def calculate(*args):
4     try:
5         value = float(feet.get())
6         meters.set((0.3048 * value * 10000.0 + 0.5)/10000.0)
7     except ValueError:
8         pass
9
10
⇒11 root = Tk()
12 root.title("Converting Feet to Meters")
13
⇒14 mainframe = ttk.Frame(root, padding="3 3 12 12")
15 mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
16 mainframe.columnconfigure(0, weight=1)
17 mainframe.rowconfigure(0, weight=1)
18
⇒19 feet = StringVar()
20 meters = StringVar()
21
22 feet_entry = ttk.Entry(mainframe, width=7, textvariable=feet)
23 feet_entry.grid(column=2, row=1, sticky=(W, E))
24
25 ttk.Label(mainframe, textvariable=meters).grid(column=2,
26     row=2, sticky=(W, E))
27 ttk.Button(mainframe, text="Calculate",
28     command=calculate).grid(column=3, row=3, sticky=W)
29
30 ttk.Label(mainframe, text="feet").grid(column=3, row=1,
31     sticky=W)
32 ttk.Label(mainframe, text="is equivalent to").grid(column=1,
33     row=2, sticky=E)

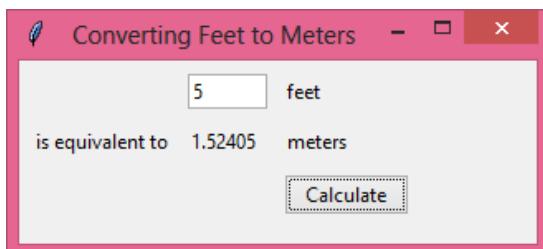
```

```

30     ttk.Label(mainframe, text="meters").grid(column=3, row=2,
31         sticky=W)
32     for child in mainframe.winfo_children():
33         child.grid_configure(padx=5, pady=5)
34     feet_entry.focus()
35     root.bind('<Return>', calculate)
36
37     root.mainloop()

```

เมื่อรันโปรแกรมจะได้ผลลัพธ์ดังรูปที่ 19.5 ให้ผู้ใช้งานใส่จำนวนฟุตที่ต้องการแปลงเป็นเมตร และกดปุ่ม calculate จะได้ผลลัพธ์ดังรูป



รูปที่ 19.5 แสดงผลการรันโปรแกรม Converting Feet to Meters

จากตัวอย่างโปรแกรมที่ 19.5 บรรทัดที่ 1 โปรแกรมทำการนำเข้าโมดูล tkinter หรือไลบรารี Tk เพื่อใช้สำหรับสร้าง GUI และบรรทัดที่ 2 เป็นคลาสที่เพิ่มเติมความสามารถให้กับ tkinter สำหรับสร้างเครื่องมือต่างๆ (themed widgets) สังเกตว่าโปรแกรมนำเข้าฟังชันและเมธอดทั้งหมดจากโมดูล tkinter โดยเรียกใช้คำสั่ง `from tkinter import *` ซึ่งส่งผลให้สามารถเรียกใช้ฟังชันและเมธอดโดยไม่ต้องอ้าง tkinter ก่อนการใช้งาน แต่สำหรับคำสั่ง `from tkinter import ttk` คือ โปรแกรมต้องการโหลดคลาสชื่อ ttk เข้ามาเสริมการทำงาน (โหลดช้อน) คลาสที่โหลดเข้ามาก่อนด้วยสัญลักษณ์ * จากบรรทัดที่ 1 นั่นเอง และต้องอ้างชื่อคลาส (ttk) ในการใช้งานก่อนเสมอ เพราะถ้าไม่อ้างชื่อ ttk ก่อน โปรแกรมจะมองว่า เรียกใช้งานเมธอดหรือคลาสจากคำสั่ง * แทนนั่นเอง บรรทัดที่ 4 สร้างเมธอดชื่อ `calculate()` เพื่อคำนวณการแปลงจากฟุตเป็นเมตร

บรรทัดที่ 11 – 17 เป็นการสร้างหน้าต่างหลัก (Main window) ชื่อ `root` โดยการหนด `title` ให้มีข้อความว่า "Converting Feet to Meters" บรรทัดที่ 14 โปรแกรมทำการสร้างเฟรม (Frame widget) ชื่อว่า `mainframe` ในหน้าต่างหลัก เพื่อรับรับเครื่องมือหรือ tools ต่างๆ ที่จะสร้างขึ้นในลำดับต่อไป

บรรทัดที่ 19 – 30 เป็นการสร้าง widgets ต่างๆ ลงบนเฟรม (mainframe) คือ Label, Entry และ Button โดยอยู่ภายใต้กริด (Grid) ที่กำหนด เช่น `feet_entry.grid(column=2, row=1, sticky=(W, E))` คือ Entry หรือช่องที่เตรียมไว้ให้ผู้ใช้งานใส่จำนวนฟุตที่ต้องการแปลง โดยวางในตำแหน่งแนวระหว่างตะวันออก (E) และตะวันตก (W)

บรรทัดที่ 32 เป็นการกำหนดให้โปรแกรมทำการรอด Widgets ต่างๆ ที่อยู่ในเฟรมลงบนกริด และบรรทัดที่ 34 โปรแกรมทำการตรวจสอบเมื่อผู้ใช้กดปุ่ม enter บนแป้นพิมพ์ (`root.bind('<Return>', calculate)`) จะทำให้โปรแกรมเรียกเมธอด `calculate()` เพื่อประมวลผลสมการแปลงหน่วยจากฟุตเป็นเมตรทันที

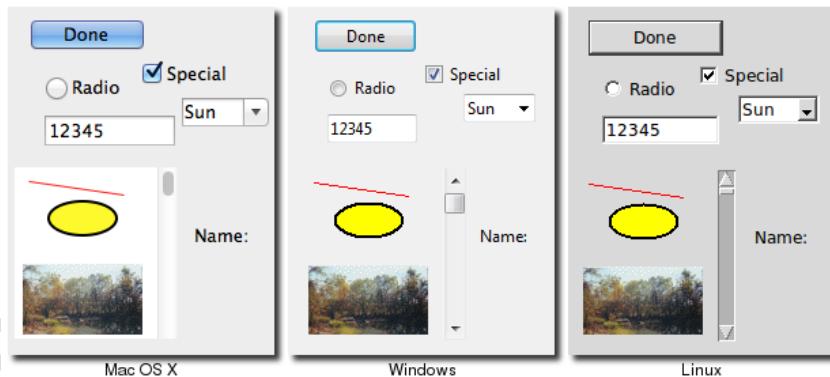
ในบรรทัดสุดท้าย (37) โปรแกรมออกให้ Tk ทำงานในลักษณะนั้นๆไปเรื่อยๆ เพื่อรับเหตุการณ์ (event) ต่างๆ ที่จะเกิดขึ้นกับโปรแกรม

2. การสร้าง GUI ด้วย Tk

ก่อนการสร้างกราฟฟิกจาก Tk หรือ Tkinter ผู้เขียนโปรแกรมจำเป็นต้องเข้าใจถึงแนวความคิดและหลักการทำงานของ Tk เสียก่อน เพื่อเป็นพื้นฐานในการสร้าง GUI ที่สมบูรณ์ต่อไป Tk ประกอบไปด้วย 3 ส่วนที่สำคัญคือ widgets, geometry management และ event handling ซึ่งมีรายละเอียดดังนี้

1. Widgets

Widgets คือสิ่งต่างๆ (บางครั้งเรียกว่า อ็อปเจกต์) ที่ปรากฏอยู่บนจอภาพ เช่น Button, Label, Frame, checkbox, tree views, scrollbars, text areas เป็นต้น ดังรูปที่ 19.6

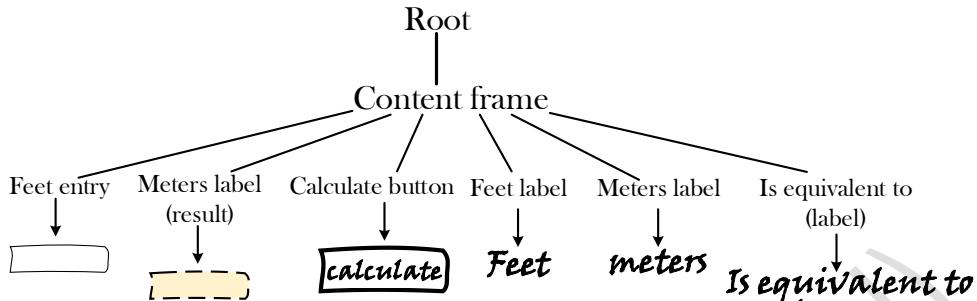


รูปที่ 19.6 แสดงตัวอย่าง Widgets บนระบบปฏิบัติการต่างๆ

อ้างอิงจาก <http://www.tkdocs.com/>

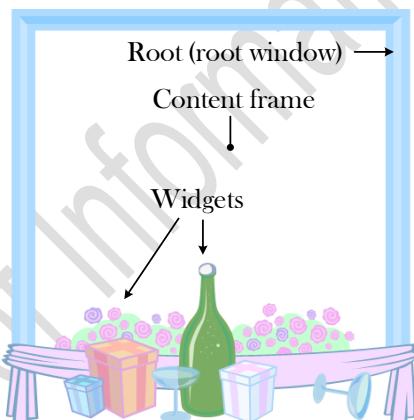
Widgets ต่างๆ เหล่านี้ถูกออกแบบให้มีลักษณะการทำงานแบบลำดับชั้น นั่นคือ การสร้าง GUI ได้ๆ ให้ปรากฏบนจอภาพ จะเริ่มต้นสร้างจากลำดับชั้นที่เรียกว่า root window (root) ขึ้นเสียก่อน ดังรูปที่ 19.7 ลำดับชั้นที่ 2 จึงสร้างเฟรม (Content frame) หรือพื้นผ้า (Canvas) เพื่อบรรจุ widgets ต่างๆ ลงบน root window ในลำดับชั้นที่ 3 เป็นการเพิ่ม Widgets ต่างๆ ที่ได้ออกแบบไว้

(บนกระดาษ) ลงบนเฟรมหรือ Canvas ที่ได้จัดเตรียมไว้ สำหรับการควบคุมการทำงานของ Widgets จะเป็นแบบลำดับชั้นเช่นเดียวกัน



รูปที่ 19.7 แสดงลำดับชั้นของการสร้าง GUI ด้วย Tk

จากรูปที่ 19.7 แสดงลำดับชั้นการสร้าง GUI ของโปรแกรมแปลงหน่วยวัดระยะทางจากฟุตเป็นเมตรที่แสดงไว้ในรูปที่ 19.4 โดยเริ่มต้นจากการสร้างหน้าต่างหลัก (Root) ขึ้นก่อนเสมอ เพื่อรองรับ Widgets ที่ทำหน้าที่ต่างๆ กัน โดย Widgets จะถูกสร้างขึ้นลงบนเฟรม (Content frame) อีกที ซึ่งเฟรมดังกล่าวเปรียบเสมือนหน้ากระดาษบนคอมพิวเตอร์นั่นเอง ซึ่งแสดงในรูปที่ 19.8



รูปที่ 19.8 แสดงตัวอย่างความสัมพันธ์ของ Root (root window), Content frame และ Widgets

2. การเรียกใช้งาน Widgets

Widgets ในภาษาไพธอนถูกเขียนขึ้นด้วยโปรแกรมเชิงวัตถุ ดังนั้นทุกๆ Widget ที่สร้างขึ้นจะถูกเรียกว่า օbjekต์ (Object) หรือวัตถุ เมื่อทำการสร้างอินสแตนซ์ของ Widget ใดๆ ขึ้นจะต้องส่งพารามิเตอร์ให้กับคลาสแม่ตามลำดับชั้นตามที่กล่าวมาแล้ว ยกเว้น Root ซึ่งเป็นคลาสแม่ที่อยู่ในตำแหน่งบนสุด (Toplevel window) ของลำดับชั้น คลาสลูกทุกๆ คลาสจะถูกสร้างภายใน root เท่านั้น จากตัวอย่างต่อไปนี้เป็นตัวอย่างการสร้าง Root, Content frame และ Widgets

```
root = Tk()      #Create root window
content = ttk.Frame(root)      #Create content frame
button = ttk.Button(content)    #Create button in frame
```

Widget แต่ละประเภทมีหน้าที่การทำงานที่ต่างกัน ดังนั้นเมื่อสร้าง Widget ขึ้นมาแล้วจำเป็นที่ต้องการทำการปรับแต่งคุณสมบัติของ Widgets (Configuration option) แต่ละตัวให้เหมาะสมกับงานที่จะทำ ตัวอย่างเช่น Label และ Button สามารถกำหนดข้อความที่แสดงผลด้วยอופชัน `text = "Text name"` ในขณะที่ scrollbar ไม่จำเป็นต้องใช้ออฟชันดังกล่าว และถ้ามีการกดปุ่ม Button จะต้องเรียกใช้พังชันหรือเมธอดที่เหมาะสมกับงานดังกล่าวโดยใช้คำสั่ง `command` แต่สำหรับ Label ไม่จำเป็นต้องใช้คำสั่ง `command` ในการทำงาน เป็นต้น



Note: ใน Python เวอร์ชันต่ำกว่า 3.0 ไฟร์อนจะใช้โมดูลชื่อ Tkinter สำหรับสร้าง GUI แต่ไฟร์อนเวอร์ชัน 3 ขึ้นไปจะเปลี่ยนชื่อเป็น tkinter และเพิ่มคลาสชื่อ tkk เข้ามาเสริมในการสร้าง GUI ดังนั้นผู้ใช้สามารถเลือกใช้คลาส Tk แบบเดิมหรือแบบใหม่ก็ได้ โดยผู้ใช้เรียกใช้คลาสที่ขึ้นต้นด้วย tkk และง่ายกว่าการใช้คลาสใหม่นั่นเอง

Configuration option จะถูกกำหนดในขณะที่ทำการสร้าง Widget โดยการส่งเป็นพารามิเตอร์ที่เหมาะสมให้กับแต่ละ Widget ที่จะถูกสร้างขึ้น ตัวอย่างเช่น เมื่อต้องการสร้างปุ่ม Button โดยปุ่มดังกล่าวประกอบด้วยชื่อของปุ่ม และเหตุการณ์ที่เกิดขึ้นหลังจากมีการกดปุ่มดังกล่าว ดังนี้

- นำเข้าโมดูล tkinter และ ttk

```
>>> from tkinter import *
>>> from tkinter import ttk
>>> root = Tk() #Create root window
```

- สร้างปุ่ม Button โดยส่งพารามิเตอร์ให้ 2 ค่าคือ ข้อความที่แสดงบนปุ่ม (`text="Hello"`) และวิธีการตอบสนองเมื่อปุ่มดังกล่าวถูกกด (`command="buttonpressed"`) พร้อมกับกำหนดค่ากริด

```
>>> button = ttk.Button(root, text="Hello",
command="buttonpressed")
>>> button.grid()
```

- แสดงข้อความบนปุ่มชื่อ button ที่กำลังทำงาน

```
>>> button['text']
'Hello'
```

- ทดสอบเปลี่ยนข้อความที่แสดงบนปุ่มจาก "Hello" เป็น "Goodbye"

```
>>> button['text']
'Hello'
>>> button['text'] = 'Goodbye'
```

```
'Goodbye'  
>>> button['text']  
'Goodbye'
```

หรือใช้เมธอด `configure()` เปลี่ยนข้อความแทนก็ได้

```
>>> button.configure(text='Goodbye')
```

- แสดงประวัติข้อมูลที่ถูกปรับปรุงแก้ไขให้กับออฟชัน 'text' ของปุ่ม button

```
>>> button.configure('text')  
('text', 'text', 'Text', '', 'goodbye')
```

- ขอดูรายละเอียดทั้งหมดของ Widget ที่ชื่อว่า Button

```
>>> button.configure()  
{'textvariable': ('textvariable', 'textVariable', 'Variable',  
'', ''), 'text': ('text', 'text', 'Text', '', 'goodbye'),  
'command': ('command', 'command', 'Command', '',  
'buttonpressed'), 'default': ('default', 'default', 'Default',  
<index object: 'normal'>, <index object: 'normal'>), 'style':  
('style', 'style', 'Style', '', ''), 'class': ('class', '',  
'', '', ''), 'state': ('state', 'state', 'State', <index  
object: 'normal'>, <index object: 'normal'>), 'compound':  
(('compound', 'compound', 'Compound', <index object: 'none'>,  
<index object: 'none'>), 'padding': ('padding', 'padding',  
'Pad', '', ''), 'takefocus': ('takefocus', 'takeFocus',  
'TakeFocus', 'ttk::takefocus', 'ttk::takefocus'), 'image':  
(('image', 'image', 'Image', '', ''), 'cursor': ('cursor',  
'cursor', 'Cursor', '', ''), 'underline': ('underline',  
'underline', 'Underline', -1, -1), 'width': ('width', 'width',  
'Width', '', ''))}
```

3. การจัดการรูปทรงเลขานุตให้กับ Widgets (Geometry management)

จากที่กล่าวมาแล้วข้างต้นว่าการวาง Widgets ลงบนเฟรมนั้น จะต้องกำหนดตำแหน่งในการวาง โดยอาศัยศาสตร์ทางด้านเลขานุตเข้าช่วย เพื่อให้ Widgets ที่วางอยู่ในตำแหน่งที่เหมาะสม ซึ่งไฟรอนมี 3 เมธอดในการจัดการเกี่ยวกับเลขานุตของ Widgets ประกอบไปด้วยเมธอด `pack()`, `grid()` และ `place()` ซึ่งจะกล่าวในหัวข้อ การจัดวาง Widgets ด้วยเลขานุต

สำหรับปัญหาระหว่าง Widgets ที่เกิดขึ้นเสมอคือ การปรับขนาดของเฟรมหรือ Widgets จะส่งผล กระทบซึ่งกันและกัน เช่น ถ้าผู้ใช้งานย่อหรือขยายขนาดของหน้าต่างหลักจะส่งผลให้กระทบกับอ้อปเจกต์ต่างๆ ที่อยู่ภายในหน้าต่างหลักนั้นๆ ทันที ซึ่งอาจจะทำให้ปุ่ม ตัวอักษร เลเบล กีดความผิดเพี้ยน ไปจากเดิม ปัญหาต่างๆ เหล่านี้จะถูกจัดการด้วย Geometry management ที่อยู่ใน Tk โดยใช้เทคนิคที่

เรียกว่า Master and Slave โดย Master คืออ็อปเจกต์ที่ทำหน้าที่รองรับ Widgets ต่างๆ ที่จะทำงาน เช่น root หรือ content frame สำหรับ Slave คือ Widgets ต่างๆ ที่วัดหรือวางแผนบน Master นั้นเอง สำหรับการทำงานของ Geometry management นั้นจะใช้วิธีสอบถามไปยัง Widgets ต่างๆ ที่กำลังจะทำงานว่าแต่ละ Widgets ต้องการพื้นที่เท่าไร ใช้สำหรับการทำงานมากน้อยเพียงใด จากนั้น Geometry management จะคำนวณพื้นที่ทั้งหมดในภาพรวมเพื่อจัดวาง Widgets เหล่านั้นในตำแหน่งที่เหมาะสม ต่อไป จากตัวอย่าง เมื่อผู้ใช้ต้องการเพิ่ม Widget อันใดอันหนึ่งลงบนเฟรมจะเรียกใช้เมธอด grid() และตามด้วยพารามิเตอร์คอลัมน์ (Column) และแถว (Row) ในการกำหนดตำแหน่งการวาง พร้อมกับพารามิเตอร์ "sticky" เพื่อบอกว่าให้วาง Widget ตามขนาดจริงที่ปรากฏ เช่น ความกว้างและยาวของปุ่ม จะมีขนาดตามฟอนต์ที่ปรากฏบนปุ่มเป็นต้น หรืออาจจะใช้เมธอด columnconfigure() หรือ rowconfigure() เพื่อปรับขนาดของ Widget ตามที่ผู้ใช้งานต้องการก็ได้

4. การจัดการกับเหตุการณ์ต่างๆ (Event Handling)

Event handling คือเหตุการณ์ต่างๆ ที่ผู้ใช้งานกระทำการกับ Widgets ใดๆ บน GUI เช่น การกดปุ่ม การกดปุ่มใดๆ บนแป้นพิมพ์ การเคลื่อนเมาส์ การปรับขนาดของหน้าต่างวินโดว์ เป็นต้น ซึ่งเหตุการณ์ต่างๆ เหล่านี้จะถูกจัดการโดย Tk ซึ่งเรียกว่า event loop โดยจะทำงานร่วมกับระบบปฏิบัติการโดยตรง เช่น เมื่อเคลื่อนเมาส์ไปยังปุ่มจะส่งผลให้ปุ่มดังกล่าวจะเปลี่ยนสีและเมื่อเคลื่อนเมาส์ออกจากปุ่มจะทำให้สีของปุ่มกลับไปเป็นสีเดิม เป็นต้น

5. การตอบสนองต่อเหตุการณ์ที่เกิดขึ้น (Command Callbacks)

มีหลาย Widgets จำเป็นต้องกระทำการต่อไปย่างใดอย่างหนึ่งเมื่อมีการคลิกหรือกระทำการกับ Widgets เหล่านั้น เช่น เมื่อกดปุ่ม Save as... จะส่งผลให้มีการเปิดหน้าต่างวินโดว์เพื่อให้ผู้ใช้เลือกไดเรกทรอรี ที่ต้องการบันทึกผลลัพธ์ดิสก์ เป็นต้น ในไฟรอนจะใช้คำสั่ง "command" หรือเรียกว่า "Callbacks" ใน การตอบสนองต่อเหตุการณ์ต่างๆ ที่เกิดขึ้นกับ Widgets โดยมีรูปแบบคำสั่งคือ

command = functionName

ตัวอย่างเช่น

```
def helloCallBack():
    print("Hello World!")
```

```
Button(mainframe, text="Hello", command=helloCallBack)
```

จากตัวอย่างโปรแกรมข้างบน เมื่อกดปุ่มชื่อว่า "Hello" โปรแกรมจะเรียกใช้ฟังชันชื่อ helloCallBack() เข้ามาทำงานแทนที่ โดยฟังชันดังกล่าวจะพิมพ์ข้อความว่า ="Hello World!"

6. การผูกเหตุการณ์ต่าง ๆ เข้ากับไฟล์

Widgets มีคำสั่ง "command" ซึ่งมาพร้อมกับขณะที่มีการสร้าง Widget อยู่แล้ว เพื่อผูกเหตุการณ์ต่าง ๆ เข้ากับการกระทำอย่างใดอย่างหนึ่งในโปรแกรม แต่ผู้ใช้สามารถเรียกใช้คำสั่ง bind เพื่อเป็นทางเลือกในการดักจับเหตุการณ์ต่าง ๆ เมื่อกับการใช้คำสั่ง command ได้เช่นเดียวกัน จากโปรแกรมตัวอย่างที่ 19.2 แสดงการจัดการกับ Widget ชนิด Label ด้วยคำสั่ง bind ดังนี้

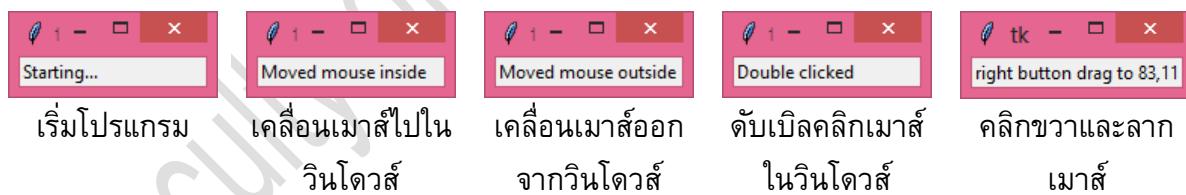
Program Example 19.2: Event Bindings (bind)

```

1  from tkinter import *
2  from tkinter import ttk
3  root = Tk()
4  l = ttk.Label(root, text="Starting...")
5  l.grid()
6  l.bind('<Enter>', lambda e: l.configure(text='Moved mouse
    inside'))
7  l.bind('<Leave>', lambda e: l.configure(text='Moved mouse
    outside'))
8  l.bind('<1>', lambda e: l.configure(text='Clicked left mouse
    button'))
9  l.bind('<Double-1>', lambda e: l.configure(text='Double
    clicked'))
10 l.bind('<B3-Motion>', lambda e: l.configure(text='right
    button drag to %d,%d' % (e.x, e.y)))
11 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่นโปรแกรมคือ



จากการด้านบนแสดงการทำงานของโปรแกรมที่ 19.2 โดยเริ่มจากบรรทัดที่ 3 โปรแกรมทำการสร้างหน้าต่างหลัก (root) โดยเรียกคลาส Tk() ขึ้นมาทำงาน จากนั้นบรรทัดที่ 4 โปรแกรมทำการสร้าง Label ลงบนหน้าต่างหลักดังกล่าว โดยมีพารามิเตอร์ 2 ตัวคือ หน้าต่างหลักที่จะเพิ่ม Label ลงไป ในที่นี่คือ root และข้อความที่จะให้ปรากฏบน Label คือ text="Starting..." บรรทัดที่ 5 โปรแกรมเรียกใช้เมธอดกริด (เมื่อไม่กำหนดคอมลัมพ์และแถวจะมีค่าเป็น 1 คอลัมน์ 1 แถว) บรรทัดที่ 6 ทำการผูกเหตุการณ์เมื่อผู้ใช้มีการกระทำใด ๆ บน Label ดังกล่าวด้วยคำสั่ง bind โดยคำสั่งนี้จะทำงานก็ต่อเมื่อผู้ใช้เคลื่อนเมาส์เข้าไปในวินโดวส์ โดยจะพิมพ์ข้อความว่า "Move mouse inside" บรรทัดที่ 7, 8, 9 และ 10 จะทำงานก็ต่อเมื่อผู้ใช้เคลื่อนเมาส์ออกจากวินโดวส์ คลิกเมาส์ซ้าย ดับเบิลคลิก และการคลิกพร้อม

กับลากเม้าส์ ตามลำดับ ผลลัพธ์แสดงดังรูปด้านบน บรรทัดที่ 11 โปรแกรมจะทำการวนลูปเพื่อรับเหตุการณ์ต่างๆ ของผู้ใช้งานไปเรื่อยๆ จนกว่าจะปิดโปรแกรม



Tips: <Enter> = เม้าส์อยู่ในขอบเขตของวินโดว์, <Leave> = นอกขอบเขตของวินโดว์,
<1> = คลิกซ้าย, <Double-1> = ดับเบิลคลิก, <B3-Motion> = คลิกขวา + ลากเม้าส์



Note: lambda คือฟังชันชนิดหนึ่งที่ไม่จำเป็นต้องประกาศชื่อฟังชัน แต่ใช้คำว่า lambda แทน เพื่อทำงานเฉพาะกิจอย่างโดยย่างหนัก (สามารถอ่านเพิ่มเติมได้ในบทที่ 7)



Note: สำหรับรหัสที่ใช้กับเหตุการณ์ต่างๆ เมื่อเกิดขึ้นกับ Widgets สามารถอ่านเพิ่มเติมได้จากเว็บไซต์ของ Tk ได้ที่ <http://www.tcl.tk/man/tcl8.5/TkCmd/bind.htm>

7. สรุปขั้นตอนการสร้าง GUI ด้วย tkinter และ tkk มีขั้นตอนดังนี้

1. นำเข้าโมดูล Tkinter (import tkinter, tkk)
2. สร้างหน้าต่าง GUI หลัก (Root window)
3. เพิ่มเครื่องมือ (Widgets) หรืออปเจกต์ที่ต้องการลงในหน้าต่างหลัก เช่น Button, Canvas หรือ Label เป็นต้น ตามหลักการของเลขาคณิต
4. เขียนโปรแกรมสำหรับตรวจสอบเหตุการณ์ต่างๆ (Events) ที่เกิดขึ้นจากผู้ใช้งาน กระทำกับอปเจกต์ใดๆ เช่น การคลิก การลาก การปล่อย เป็นต้น
5. ส่งให้โปรแกรมวนลูปรับคำสั่งจากผู้ใช้งานไปเรื่อยๆ จนกว่าโปรแกรมจะยุติการทำงาน

ตัวอย่างโปรแกรมที่ 19.3 แสดงตัวอย่างลำดับการสร้างหน้าต่าง GUI ด้วย Tk

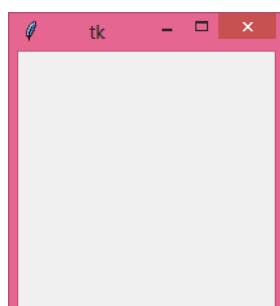
Program Example 19.3: Step of creating GUI by Tk

```

⇒1 import tkinter
⇒2 root = tkinter.Tk()
⇒3 # Code to add widgets will go here...
⇒4 root.mainloop()

```

ผลลัพธ์แสดงดังรูปที่ 19.9



รูปที่ 19.9 แสดงหน้าต่างหลัก (root window)

จากตัวอย่างโปรแกรมที่ 19.3 บรรทัดที่ 1 เป็นการนำเข้าโมดูล tkinter เพื่อใช้สำหรับสร้างกราฟฟิก บรรทัดที่ 2 เรียกใช้งานคลาส Tk() เพื่อสร้างอ็อกเจ็คต์ของหน้าต่าง GUI หลัก ค่าอ้อมเปจект์ของหน้าต่าง หลักที่ได้จะถูกไว้ในตัวแปรชื่อ root ในบรรทัดที่ 3 เป็นตำแหน่งของโปรแกรมที่ใช้สำหรับเพิ่ม Widgets ต่างๆ ที่ผู้เขียนโปรแกรมต้องการ (ในตัวอย่างนี้ยังไม่มีการเพิ่ม Widgets ใดๆ) บรรทัดสุดท้ายเรียกใช้ เมธอด mainloop() เพื่อใช้สำหรับควบคุมและดักจับเหตุการณ์ต่างๆ ที่ผู้ใช้กระทำบนหน้าต่าง GUI หลัก

3. การจัดวาง Widgets ด้วยเลขาคณิต (Geometry management)

จากที่กล่าวมาแล้วข้างต้นว่าทุกๆ Widgets จะต้องอาศัยเลขาคณิตช่วยในการจัดวาง (Geometry management) ซึ่งไฟรอนได้จัดเตรียมไว้ 3 เมธอดคือ pack(), grid() และ place() ดังนี้

- เมธอด **pack()** ทำหน้าที่จัดวาง Widgets ให้อยู่ในกลุ่ม (block) ก่อนวางลงใน Widget แม่ ซึ่งมีรูปแบบคำสั่งดังนี้

```
widget.pack( pack_options )
```

pack_options มี 3 รูปแบบคือ expand, fill และ side

- expand: เมื่อกำหนดให้เป็น True หรือ YES โปรแกรมจะขยายขนาด Widget โดยการเพิ่ม ซองว่างเข้าไปแทน และอยู่ตรงกลางหน้าต่างหลัก เช่น `widget.pack(expand=True)`



- fill: กำหนดให้ขยายขนาดของ Widget ตามขนาดจริงที่น้อยที่สุด หรือสามารถกำหนดเป็น Y (vertically: ขยายทางแนวตั้ง) หรือกำหนดเป็น X (horizontally: ขยายทางแนวนอน) หรือทั้งคู่ก็ได้ แต่ค่าดีฟอลต์ (default) เป็น NONE เช่น `fill = X`, `fill = Y` หรือ `fill = BOTH` เช่น `widget.pack(fill=BOTH)`



- side: กำหนดตำแหน่งในการวาง Widget คือ TOP (ดีฟอลต์) ด้านบน, BOTTOM ด้านล่าง, LEFT ด้านซ้าย และ RIGHT ด้านขวา เช่น `side = BOTTOM` เช่น `widget.pack(side=BOTTOM)`



สำหรับตัวอย่างการใช้งานเมธอด pack() ดังนี้

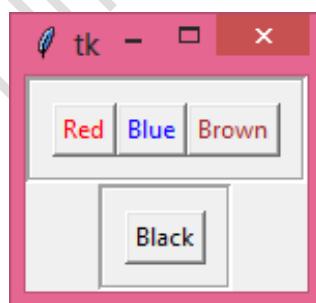
Program Example 19.4: *pack()* method

```

1  from tkinter import *
2
3  root = Tk()
4  frame = Frame(root, bd="3", relief=GROOVE, padx=10, pady=10)
5  frame.pack()
6  redbutton = Button(frame, text="Red", fg="red")
7  redbutton.pack(side = LEFT)
8  greenbutton = Button(frame, text="Brown", fg="brown")
9  greenbutton.pack(side = RIGHT)
10 bluebutton = Button(frame, text="Blue", fg="blue")
11 bluebutton.pack(side = BOTTOM)
12
13 bottomframe = Frame(root, bd="3", relief=GROOVE, padx=10,
14     pady=10)
15 bottomframe.pack(side = BOTTOM)
16 blackbutton = Button(bottomframe, text="Black", fg="black")
17 blackbutton.pack(side = BOTTOM)
18 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูป



จากตัวอย่างโปรแกรมที่ 19.4 แสดงการใช้เมธอด pack() ในการจัดวาง Widgets โดยโปรแกรมสร้างเฟรม (บรรทัดที่ 4) ด้วยเมธอด Frame(root) อ้อปเจ็กต์ที่สร้างขึ้นจะเก็บไว้ในตัวแปรชื่อ frame จากนั้นบรรทัดที่ 5 โปรแกรมเรียกเมธอด pack() โดยไม่มีพารามิเตอร์ ซึ่งส่งผลให้เฟรมดังกล่าวจะวางอยู่ในตำแหน่งบนสุดของหน้าต่างหลัก (root window) บรรทัดที่ 6 – 11 โปรแกรมสร้างปุ่มสีแดง นำเงิน และนำตาลงบนเฟรมดังกล่าว

บรรทัดที่ 13 และ 14 โปรแกรมสร้างเฟรมชื่อ bottomframe โดยวางอยู่ในตำแหน่งด้านล่างของหน้าต่างหลัก บรรทัดที่ 15 และ 16 โปรแกรมสร้างปุ่มสีดำและวางลงในเฟรมชื่อ bottomframe

2. เมธอด **grid()** ทำหน้าที่จัดวาง Widgets ลงในหน้าต่างหลัก (root window) โดยอยู่ในรูปแบบของตาราง ซึ่งมีรูปแบบคำสั่งดังนี้

```
widget.grid( grid_options )
```

grid_options มีรูปแบบดังนี้คือ

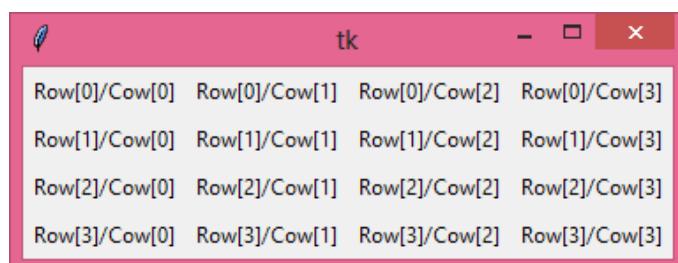
- column: ตำแหน่งคอลัมน์ที่ต้องการวาง Widget ค่าดีฟอลต์คือ 0 (คอลัมน์ช้ายสุด)
- colspan: จำนวนคอลัมน์ที่ต้องการใช้ ค่าดีฟอลต์คือ 1 บรรทัด
- ipadx, ipady: ขนาดภายในขอบของ Widget ในแนวตั้งและแนวนอน มีหน่วยเป็นพิกเซล
- padx, pady: ขนาดภายนอกขอบของ Widget ในแนวตั้งและแนวนอน มีหน่วยเป็นพิกเซล
- row: ตำแหน่งแถวที่ต้องการวาง Widget
- rowspan: จำนวนแถวที่ต้องการใช้ ค่าดีฟอลต์คือ 1 บรรทัด
- sticky: ใช้มื่อต้องการวาง Widget ในตำแหน่งที่ต้องการ โดยค่าดีฟอลต์จะอยู่ตรงกลาง แต่สามารถกำหนดตำแหน่งด้วยการบอกทิศทาง โดยใช้อักษรต่างๆ ดังนี้ N (ทิศเหนือ), E (ตะวันออก), S (ใต้), W (ตะวันตก), NE (ทิศตะวันออกเฉียงเหนือ), NW (ทิศตะวันตกเฉียงเหนือ), SE (ทิศตะวันออกเฉียงใต้) และ SW (ทิศตะวันตกเฉียงใต้) เป็นต้น

สำหรับตัวอย่างการใช้งานเมธอด **grid()** ดังนี้

Program Example 19.5: *grid() method*

```
1 import tkinter
2 root = tkinter.Tk( )
3 for r in range(4):
4     for c in range(4):
5         tkinter.Label(root, text='Row[%s]/Cow[%s]'%(r,c),
6 borderwidth=5).grid(row=r,column=c)
7 root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูป



จากตัวอย่างโปรแกรมที่ 19.5 แสดงการใช้เมธอด grid() ในการจัดวาง Widgets โดยบรรทัดที่ 5 โปรแกรมสร้างเลเบลที่ทำการพิมพ์ตำแหน่งແກ່ວແລະຄອລັນໜົງບນວນໂດສວ່ຫລັກ ໂດຍໃຊ້ເມືອດກຣິດໃນການ ກຳນົດຕຳແໜ່ງຂອງເບີເລ ພລັບພົມທີ່ໄດ້ແສດງດັ່ງນີ້ປັດ້ນນ

3. เมธอด place() ທຳກັນທີ່ຈັດວາງ Widgets ລົງໃນໜ້າຕ່າງໆ (root window) ໂດຍການ ຮະບຸຕຳແໜ່ງ ຊຶ່ງມີຮູບແບບຄຳສັ່ງດັ່ງນີ້

```
widget.place( place_options )  
place_options ມີຮູບແບບດັ່ງນີ້ຄືວ່າ
```

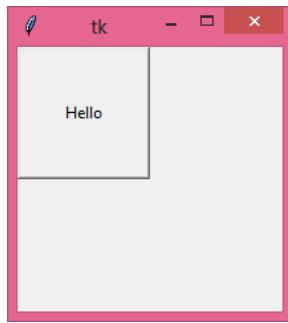
- anchor: ກຳນົດຕຳແໜ່ງກາງ Widget ໂດຍອາຄັຍທີ່ ເຊັ່ນ N (ທີ່ເກີດເກີດ), E (ຕະວັນອອກ), S, W, NE, NW, SE ອີ່ວີ່ SW ເປັນຕົ້ນ
- bordermode: ກຳນົດຕຳແໜ່ງກາງ Widget ໂດຍອາຄັຍຂອບດ້ານໃໝ່ (INSIDE) ແລະຂອບ ດ້ານອອກ (OUTSIDE) ດ້ວຍຕົວຢ່າງ INSIDE
- height, width: ກຳນົດຂາດຄວາມກວ້າງແລະຄວາມຍາວ ມີໜ່ວຍເປັນພິກເໜ້ນ
- relheight, relwidth: ກຳນົດຂາດຄວາມກວ້າງແລະຄວາມຍາວ ໂດຍໃຊ້ເລີຂໍຈຳນວນຈິງຮ່ວ່າງ 0.0 – 1.0
- relx, rely: ຂາດແນວຕັ້ງແລະແນວອນຂອງ offset ໂດຍໃຊ້ເລີຂໍຈຳນວນຈິງຮ່ວ່າງ 0.0 – 1.0
- x, y: ຂາດແນວຕັ້ງແລະແນວອນຂອງ offset ມີໜ່ວຍເປັນພິກເໜ້ນ

ສໍາຫຼັບຕົວຢ່າງການໃໝ່ງມີເມືອດ place() ຕັ້ງນີ້

Program Example 19.6: place() method

```
1 from tkinter import *  
2 import tkinter  
3  
⇒4 root = tkinter.Tk()  
5  
⇒6 def helloCallBack():  
7     tkinter.messagebox.showinfo("Hello Python", "Hello World")  
8  
⇒9 B = tkinter.Button(root, text ="Hello", command =  
helloCallBack)  
10 B.pack()  
⇒11 B.place(bordermode=OUTSIDE, height=100, width=100)  
12  
13 root.mainloop()
```

ພລັບພົມທີ່ໄດ້ເນື້ອສັ່ງຮັນໂປຣແກຣມດັ່ງນີ້ປັດ້ນລ່າງ



จากตัวอย่างโปรแกรมที่ 19.6 แสดงการใช้เมธอด `place()` ในการจัดวาง Widgets โดยบรรทัดที่ 4 โปรแกรมสร้างหน้าต่างหลัก โดยอ้างชื่อโมดูลและตามด้วยคลาส Tk (`tkinter.Tk()`) ซึ่งเป็นวิธีการสร้างหน้าต่างหลักอีกรูปแบบหนึ่ง บรรทัดที่ 6 โปรแกรมสร้างเมธอดชื่อว่า `helloCallBack()` ทำหน้าที่แสดงกล่องข้อความ (messagebox) โดยพิมพ์ข้อความว่า "Hello World" เมธอดดังกล่าวนี้จะถูกเรียกใช้เมื่อมีการกดปุ่ม บรรทัดที่ 9 โปรแกรมสร้างปุ่มซึ่งมีข้อความบนปุ่มคือ "Hello" โดยมีพารามิเตอร์ 3 ตัวคือ ชื่อปุ่มถูกกด ในที่นี้โปรแกรมจะเรียกเมธอด `helloCallBack()` มาทำงาน บรรทัดที่ 11 โปรแกรมจะวางปุ่มดังกล่าวบนขอบ (border=OUTSIDE) ที่เมธอด `pack()` ได้กำหนดไว้ โดยปุ่มนี้ขนาดความกว้างและความยาวเท่ากับ 100 พิกเซล

4. คุณสมบัติพื้นฐานของ Widgets

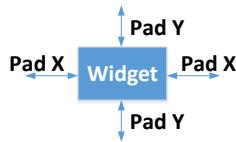
Widgets ต่างๆ ที่ใช้สำหรับออกแบบ GUI มีคุณสมบัติพื้นฐานหลายประการที่สามารถใช้งานร่วมกันได้ เช่น ขนาด สี และฟอนต์ เป็นต้น ซึ่งมีรายละเอียดของคุณสมบัติต่างๆ ดังนี้

1. Dimension (ขนาด): ขนาดของ Widgets สามารถกำหนดได้หลายแบบดังนี้

- กำหนดขนาดด้วยตัวอักษร เช่น 'c' = เซนติเมตร (Centimeters), 'i' = นิ้ว (Inches), 'm' = มิลลิเมตร (Millimeters), 'p' = Printer's points (ประมาณ 1/72)
- กำหนดขนาดด้วยตัวเลข มีหน่วยเป็นพิกเซล (Pixels) เช่น 1, 3, หรือ 5 เป็นต้น

Dimension ถูกนำไปใช้กำหนดคุณสมบัติของ Widgets ดังต่อไปนี้

- `borderwidth`: ความกว้างเส้นขอบของ Widget
- `highlightthickness`: ความกว้างของรูปสี่เหลี่ยมที่ใช้สำหรับเน้นความสนใจ (highlight rectangle) เมื่อ Widget ถูกโฟกัส
- `padX padY`: ขนาดพื้นที่ว่างเพิ่มเติมรอบๆ Widget



- `selectborderwidth`: ความกว้างเส้นขอบเมื่อ Widget ถูกเลือก
- `wraplength`: ความยาวสูงสุดที่เกิดจากการครอบคำหรือข้อความ
- `height`: กำหนดความสูงของ Widget
- `width`: ความกว้างของ Widget
- `underline`: ชี้เด่นใต้ตัวอักษรที่ต้องการ (0 คืออักษรตัวแรกของข้อความ)

2. Color (สี): การกำหนดสีให้กับตัวอักษรหรือ Widgets ได้ 2 รูปแบบคือ

- กำหนดค่าของสีโดยใช้เลขฐาน 16 ตัวอย่างเช่น `#ffff` คือสีขาว, `#0000` คือสีดำ หรือ `#00ffff` คือสีฟ้า เป็นต้น
- กำหนดค่าของสีโดยใช้ชื่อความ เช่น "red" = สีแดง, "green" = สีเขียว หรือ "white" = สีขาว เป็นต้น ดังรูปด้านล่าง



อ้างอิงจาก: <http://wiki.tcl.tk/37701>

Color ถูกนำไปใช้กำหนดคุณสมบัติของ Widgets ดังต่อไปนี้

- activebackground: กำหนดสีพื้นหลังของ Widgets เมื่อ Widgets ทำงาน (Active)
- activeforeground: กำหนดสีด้านหน้าของ Widgets เมื่อ Widgets ทำงาน (Active)
- background: กำหนดสีพื้นหลังของ Widgets หรือเขียนย่อเป็น bg
- foreground: กำหนดสีด้านหน้าของ Widgets หรือเขียนย่อเป็น fg
- disabledforeground: กำหนดสีด้านหน้าของ Widgets เมื่อ Widgets ถูก Disable
- highlightbackground: กำหนดสีพื้นหลังเมื่อ Widgets ถูกโฟกัส
- highlightcolor: กำหนดสีด้านหน้าเมื่อ Widgets ถูกโฟกัส
- selectbackground: กำหนดสีพื้นหลังเมื่อมีการเลือกรายการในรายการหนึ่งใน Widget เช่น เลือกรายการจาก Dropdown
- selectforeground: กำหนดสีด้านหน้าเมื่อมีการเลือกรายการในรายการหนึ่งใน Widget

3. Font (ฟอนต์): การกำหนดรูปแบบของฟอนต์ให้กับ Widgets สามารถกำหนดได้ 2 รูปแบบคือ

- กำหนดฟอนต์โดยใช้เครื่องหมายว่างเล็บครอบ เช่น ("Helvetica", "16") หรือ ("Times", "24", "bold italic")
- กำหนดฟอนต์โดยใช้เมธอด font ที่มากับ Tk ซึ่งมีรูปแบบคือ

```
myFont = font.Font( option, ... )
```

ตัวอย่างเช่น

```
from tkinter import *
from tkinter import font
root = Tk()
myFont = font.Font(family='Helvetica', size=12,
weight='bold')
```

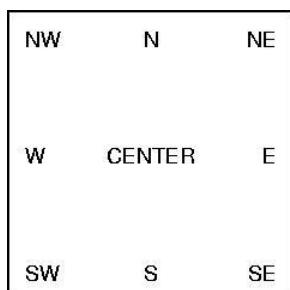
สำหรับ option มีรายละเอียดดังนี้

- family: ชื่อของฟอนต์ เช่น 'Helvetica', 'Verdana', 'Times' ตัวอย่าง family = 'Helvetica'
- size: ขนาดของฟอนต์ เช่น size = 15
- weight: ขนาดความหนาของฟอนต์ เช่น bold คือฟอนต์ตัวหนา, normal คือ ความหนาปกติ ตัวอย่าง weight = 'bold'
- slant: ตัวอักษรเอียง เช่น italic คือตัวอักษรเอียง, roman คือตัวอักษรปกติ
- underline: ขีดเส้นใต้ เช่น underline = 1 คือการขีดเส้นตัวอักษร, 0 คือ ตัวอักษรปกติ

- overstrike: ขีดเส้นทับตัวอักษร เช่น overstrike=1 คือขีดเส้นทับตัวอักษร, 0 คือไม่ขีดเส้นทับ

4. Anchors (แองเคอร์): กำหนดจุดอ้างอิงที่ใช้สำหรับจัดวาง Widgets มีรูปแบบคือ NW: ทิศตะวันตกเฉียงเหนือ, N: ทิศเหนือ, NE: ทิศตะวันออกเฉียงเหนือ, W: ทิศตะวันตก, CENTER: จุดกลางของแผนที่, E: ทิศตะวันออก, SW: ทิศตะวันตกเฉียงใต้, S: ทิศใต้, SE: ทิศตะวันออกเฉียงใต้

ตัวอย่างเช่น anchor = NE, anchor = SE สำหรับตัวอย่างตำแหน่งทิศต่างๆ แสดงดังรูปที่ 19.10



รูปที่ 19.10 แสดงตำแหน่งทิศต่างๆ ที่ใช้กับ anchor

5. Relief styles: กำหนดลักษณะภาพในรูปแบบ 3 มิติ (3-D) รอบๆ บริเวณ Widgets มีรูปแบบคือ

- FLAT: แสดงภาพในลักษณะแบบราบ (ไม่เป็นภาพมุน)
- RAISED: แสดงภาพในลักษณะยกขึ้น หรืออนุนขึ้น
- SUNKEN: แสดงภาพในลักษณะกลบลงไป
- GROOVE: แสดงภาพในลักษณะกรอบเป็นร่องลึก
- RIDGE: แสดงภาพในลักษณะกรอบมุนขึ้น

สำหรับตัวอย่างการใช้งาน Relief และแสดงในโปรแกรมตัวอย่างที่ 19.7 ดังนี้

Program Example 19.7: Relief styles

```

1 from tkinter import *
2 import tkinter
3
4 root = tkinter.Tk()
5
6 B1 = tkinter.Button(root, text ="FLAT", relief=FLAT)
7 B2 = tkinter.Button(root, text ="RAISED", relief=RAISED)
8 B3 = tkinter.Button(root, text ="SUNKEN", relief=SUNKEN)
9 B4 = tkinter.Button(root, text ="GROOVE", relief=GROOVE)
10 B5 = tkinter.Button(root, text ="RIDGE", relief=RIDGE)

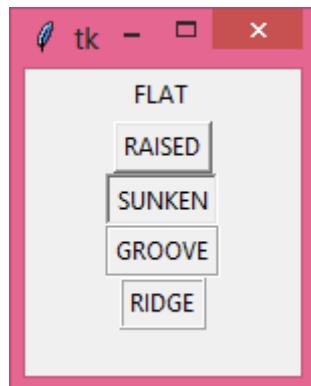
```

```

11
12 B1.pack()
13 B2.pack()
14 B3.pack()
15 B4.pack()
16 B5.pack()
17 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



6. Bitmaps (บิตแมป)

Bitmap คือภาพที่เกิดจากจุดสีที่เรียกว่า Pixel (พิกเซล) ประกอบกันเป็นรูปร่างบนพื้นที่ที่มีลักษณะเป็นเส้นตาราง (กริด) แต่ละพิกเซลจะมีค่าของตำแหน่ง และค่าสีของตัวเอง ภาพหนึ่งภาพจะประกอบด้วยพิกเซลหลาย ๆ พิกเซลผสมกัน ไฟล์ภาพเหล่านี้มีหลายรูปแบบ อาทิ เช่น BMP, TIF, JPG, PCT เป็นต้น ไฟล์อนุมีภาพบิตแมปให้เลือกใช้งานดังรูปที่ 19.11



รูปที่ 19.11 แสดงภาพบิตแมปที่ใช้งานร่วมกับ Widgets

สำหรับตัวอย่างการใช้งาน Bitmaps แสดงในโปรแกรมตัวอย่างที่ 19.8 ดังนี้

Program Example 19.8: Bitmaps

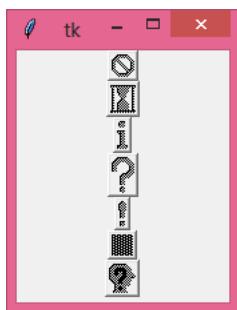
```

1 from tkinter import *
2 import tkinter
3
4 root = tkinter.Tk()
5 B1 = tkinter.Button(root, relief=RAISED, bitmap="error")
6 B2 = tkinter.Button(root, relief=RAISED, bitmap="hourglass")
7 B3 = tkinter.Button(root, relief=RAISED, bitmap="info")

```

```
8  B4 = tkinter.Button(root, relief=RAISED, bitmap="question")
9  B5 = tkinter.Button(root, relief=RAISED, bitmap="warning")
10 B6 = tkinter.Button(root, relief=RAISED, bitmap="gray75")
11 B7 = tkinter.Button(root, relief=RAISED, bitmap="questhead")
12 B1.pack()
13 B2.pack()
14 B3.pack()
15 B4.pack()
16 B5.pack()
17 B6.pack()
18 B7.pack()
19 root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



7. **Cursors (เคอร์เซอร์)**: เคอร์เซอร์หรือตัวชี้เม้าส์ คือ สัญลักษณ์แสดงตำแหน่งของเม้าส์ บนจอภาพ ไฟชอนเตรียมสัญลักษณ์สำหรับใช้เป็นเคอร์เซอร์ให้เลือกใช้งานดังรูปที่

19.12

X cursor	dotbox	man	sizing
arrow	double arrow	middlebutton	spider
based arrow down	draft large	mouse	spraycan
based arrow up	draft small	pencil	star
boat	draped box	pirate	target
bogosity	exchange	plus	tcross
bottom left corner	fleur	question arrow	top left arrow
bottom right corner	gobbler	right ptr	top left corner
bottom side	gumby	right side	top right corner
bottom tee	hand1	right tee	top side
box spiral	hand2	rightbutton	top tee
center ptr	heart	rtl logo	trek
circle	icon	sailboat	ul angle
clock	iron cross	sb down arrow	umbrella
coffee mug	left ptr	sb h double arrow	ur angle
cross	left side	sb left arrow	watch
cross reverse	left tee	sb right arrow	xterm
crosshair	leftbutton	sb up arrow	
diamond cross	ll angle	sb v double arrow	
dot	lr angle	shuttle	

รูปที่ 19.12 แสดงสัญลักษณ์ต่างๆ ที่ใช้เป็นเคอร์เซอร์

สำหรับตัวอย่างการใช้งานเคอร์เซอร์ แสดงในโปรแกรมตัวอย่างที่ 19.9 ดังนี้

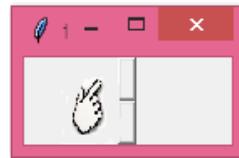
Program Example 19.9: Cursors

```

1 from tkinter import *
2 import tkinter
3
4 root = tkinter.Tk()
5
6 B1 = tkinter.Button(root, relief=RAISED, cursor="hand1")
7 B2 = tkinter.Button(root, relief=RAISED, cursor="heart")
8 B1.pack()
9 B2.pack()
10 root.mainloop()

```

ผลลัพธ์ที่ได้มีอธิบายในโปรแกรมดังรูปด้านล่าง ให้ทดสอบเลือกเมาส์ไปทับที่ปุ่มดังกล่าว

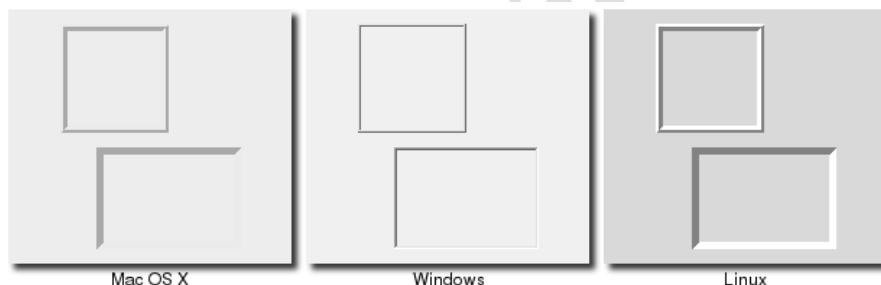


5. การสร้างและใช้งาน Widgets พื้นฐาน

ในหัวข้อนี้จะกล่าวถึงวิธีการสร้าง Widgets พื้นฐานที่สำคัญๆ สำหรับใช้ในการสร้าง GUI เช่น frames, labels, buttons, checkbuttons, radiobuttons, entries และ comboboxes เป็นต้น ซึ่งผู้เขียนแนะนำให้อ่านไปตามลำดับ เพราะเนื้อหาทั้งหมดจะมีความเกี่ยวเนื่องกัน

1. Frame (เฟรม)

เฟรมเป็น Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยม โดยปกติเฟรมจะถูกใช้สำหรับจัดกลุ่ม หรือบรรจุ Widgets อื่นๆ ที่เกี่ยวข้องหรือสัมพันธ์กันเข้าไว้ด้วยกัน รูปร่างของเฟรมแสดงในรูปที่ 19.13



รูปที่ 19.13 แสดงรูปแบบของเฟรมบนระบบปฏิบัติการต่างๆ

อ้างอิงจาก <http://www.tkdocs.com/>

รูปแบบคำสั่งสำหรับการสร้างเฟรมคือ

```
f = Frame( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของเฟรม แสดงในตารางด้านล่าง

Option	คำอธิบาย
bd	 กำหนดขนาดความกว้างของขอบเฟรมมีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Frame(root, bd=5, height=50, width=100, relief=GROOVE)

bg		กำหนดสีพื้นด้านหลังของเฟรม เช่น Frame(root, bd=5, height=50, width=100, relief=GROOVE, bg="green")
cursor		กำหนดรูปแบบของเมาส์ เคอร์เซอร์จะเปลี่ยนรูปเมื่อเคลื่อนเมาส์ทับบนเฟรม เช่น Frame(root, bd=3, height=50, width=100, relief=GROOVE, cursor="hand1")
height		กำหนดความสูงของเฟรมมีหน่วยเป็นพิกเซล เช่น Frame(root, bd=3, height=50, width=100, relief=GROOVE)
width		กำหนดความกว้างของเฟรม ถ้าไม่กำหนดให้หอนจะกำหนดขนาดเท่ากับความกว้างของฟอนต์แทน เช่น Frame(root, bd=3, height=50, width=100)
highlightbackground		กำหนดແຄບສື່ພື້ນหลังเมื่อเฟรมได้รับความสนใจ (Focus) เช่น Frame(root, bd=3, height=50, width=100, highlightbackground="green")
highlightcolor		กำหนดແຄບສື່ເມື່ອເຝັ້ນໄດ້ຮັບຄວາມສົນໃຈ เช่น Frame(root, bd=3, height=50, width=100, highlightcolor="green")
highlightthickness		กำหนดความกว้างจากขอบของເຝັ້ນ เช่น Frame(root, bd=3, height=50, width=100, highlightthickness =2)
relief		กำหนดลักษณะເຝັ້ນໃນรูปແບບ 3 ມີດີ เช่น Frame(root, bd=5, height=50, width=100, relief=GROOVE)

สำหรับตัวอย่างการใช้งานเฟรม แสดงในโปรแกรมตัวอย่างที่ 19.10 ดังนี้

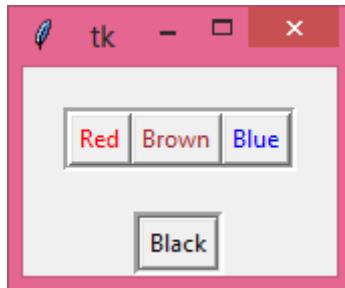
Program Example 19.10: Frame and pack()

```

1  from tkinter import *
2
3  root = Tk()
4  frame = Frame(root, bd=3, height=300, width=500,
5    relief=GROOVE, cursor="hand1", highlightthickness=20)
6  frame.pack(expand=True)
7  redbutton = Button(frame, text="Red", fg="red")
8  redbutton.pack(side=LEFT)
9  greenbutton = Button(frame, text="Brown", fg="brown")
10 greenbutton.pack(side=LEFT)
11 bluebutton = Button(frame, text="Blue", fg="blue")
12 bluebutton.pack(side=LEFT)
13
14 bottomframe = Frame(root, bd=3, cursor="hand1",
15   relief=SUNKEN)
16 bottomframe.pack(side = BOTTOM)
17 blackbutton = Button(bottomframe, text="Black", fg="black")
18 blackbutton.pack(side = BOTTOM)

```

18 | root.mainloop()
ผลลัพธ์ที่ได้เมื่อสิ่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.10 แสดงการสร้างเฟรมเพื่อรับรับการวาง Widget ชนิดปุ่มลงในเฟรมดังกล่าว บรรทัดที่ 4 เป็นการสร้างเฟรมชื่อ frame ให้มีขนาดสูงเท่ากับ 300 และกว้างเท่ากับ 500 โดยมีความหนาของขอบเฟรมเท่ากับ 3 (bd=3) กรอบของเฟรมเป็นแบบร่องลึก (relief=GROOVE) มีขนาดความกว้างจากขอบเฟรมเท่ากับ 20 (highlightthickness=20) เมื่อเลื่อนเมาส์เข้าไปเฟรมดังกล่าว เคอร์เซอร์จะเปลี่ยนเป็นรูปมือ (cursor="hand1") และเฟรมที่สร้างขึ้นจะเขียนลงบนหน้าต่างหลักของโปรแกรม (root) บรรทัดที่ 5 เป็นการกำหนดให้เฟรมดังกล่าววางลงตรงกลางหน้าต่างหลักด้วยเมธอด frame.pack(expand=True) บรรทัดที่ 6 – 11 โปรแกรมทำการสร้างปุ่ม (ซึ่งจะกล่าวในหัวข้อด้านไป) มีข้อความสีแดง นำตาล และนำเงินลงบนเฟรมดังกล่าว ตามลำดับ สังเกตว่าขนาดของเฟรมจะเท่ากับขนาดของปุ่มเนื่องจากเมธอด pack() จะบีบอัดให้เฟรมมีขนาดเท่ากับผลรวมความกว้างและความยาวของ Widgets ที่อยู่ในเฟรมนั้นเอง (โปรแกรมจะไม่สนใจความสูงและความกว้างที่กำหนดในเฟรม)

บรรทัดที่ 13 โปรแกรมทำการสร้างเฟรมอีกรame ชื่อ bottomframe โดยเฟรมดังกล่าวมีลักษณะของกรอบที่จะลึกลงไป (relief=SUNKEN) และเฟรมดังกล่าวถูกจัดวางให้อยู่ในตำแหน่งด้านล่างของหน้าต่างหลักด้วยเมธอด bottomframe.pack(side = BOTTOM) แสดงในบรรทัดที่ 14 จากนั้นโปรแกรมสร้างปุ่มที่มีข้อความสีดำ ใส่ลงในเฟรม bottomfram โดยกำหนดให้วางในตำแหน่งด้านล่างของเฟรม (side = BOTTOM) ผลลัพธ์ที่ได้แสดงในรูปด้านบน ผู้เขียนโปรแกรมสามารถเลือกใช้เมธอด grid() แทนเมธอด pack() ในการจัดวาง Widgets ก็ได้ ดังตัวอย่างโปรแกรมที่ 19.11

Program Example 19.11: Frame and Grid()

```

1 from tkinter import *
2
3 root = Tk()
4 frame = Frame(root, bd=3, height=300, width=500,
5 relief=GROOVE, cursor="hand1", highlightthickness=20)
6 frame.pack()
7
8 Button(frame, text="Red", fg="red").grid(column=2, row=3)
9 Label(frame, text=" ", fg="red").grid(column=3, row=3)
10 Button(frame, text="Brown", fg="brown").grid(column=4, row=3)
11 Label(frame, text=" ", fg="red").grid(column=5, row=3)
12 Button(frame, text="Blue", fg="blue").grid(column=6, row=3)

```

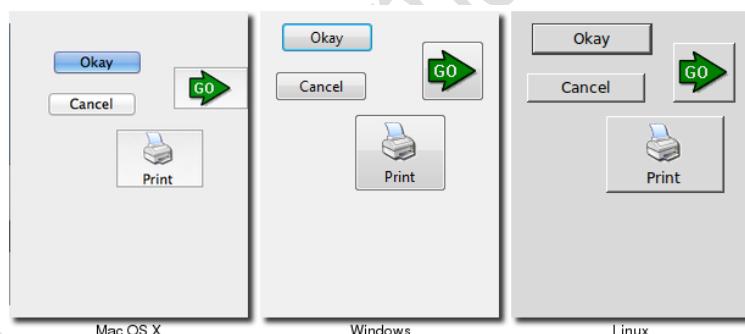
```
12 |  
13 | root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



2. Button (ปุ่ม)

Button คือ Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยมผืนผ้า มีหน้าที่ตอบสนองกับผู้ใช้งาน โดยวิธีการกดลงและปล่อย การแสดงผลบนปุ่มจะเป็นได้ทั้งข้อความหรือรูปภาพก็ได้ เมื่อกดปุ่ม และปล่อย (เกิด event) จะส่งผลให้เกิดการเรียกใช้งานฟังชันหรือเมธอดที่ผูกอยู่กับปุ่มได้ เพื่อ ทำหน้าที่อย่างใดอย่างหนึ่ง เช่น กดปุ่ม Cancel จะทำให้โปรแกรมยกเลิกคำสั่งที่เพิ่งกระทำเสร็จ เป็นต้น รูป่างของปุ่มแสดงในรูปที่ 19.14



รูปที่ 19.14 แสดงรูปแบบของปุ่มบนระบบปฏิบัติการต่างๆ

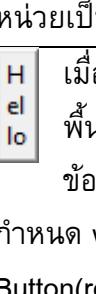
รูปแบบคำสั่งสำหรับการสร้างปุ่มคือ

`b = Button(root, option=value, ...)`

- `root` คือ วินโดว์หลัก (root window)
- `option` คือ คุณสมบัติต่างๆ ของปุ่ม แสดงในตารางด้านล่าง

Option	คำอธิบาย
<code>activebackground</code>	<code>Hello</code> กำหนดสีบนพื้นหลังของปุ่มเมื่อผู้ใช้คลิกบนปุ่ม เช่น <code>Button(root, text ="Hello", activebackground="yellow")</code>

activeforeground		กำหนดสีข้อความบนปุ่มเมื่อผู้ใช้คลิกบนปุ่ม เช่น Button(root, text = "Hello", activebackground="yellow", activeforeground="red")
bd		กำหนดขนาดความกว้างของปุ่มเมื่อห่วงเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Button(root, text = "Hello", bd=5)
bg		กำหนดสีพื้นหลังของปุ่ม เช่น Button(root, text = "Hello", bg="red")
command		ฟังชันหรือเมธอดที่จะถูกเรียกใช้เมื่อปุ่มถูกกดหรือคลิก เช่น Button(root, text = "Hello", bg="red", command=myFunc)
fg		กำหนดสีของฟอนต์ เช่น Button(root, text = "Hello", bg="red", fg="blue")
font		กำหนดรูปแบบฟอนต์ของปุ่ม เช่น Button(root, text = "Hello", bg="red", font="Times 10 bold") หรือ font=("Helvetica", 16)
height		กำหนดความสูงของปุ่ม (กรณีปุ่มที่ถูกสร้างด้วยรูปภาพจะมีหน่วยเป็น พิกเซล) เช่น Button(root, text = "Hello", height=5)
highlightcolor		กำหนดແຕບສีມีເຟຣີມໄດ້ຮັບຄວາມສຳໃຈ เช่น Button(root, text = "Hello", highlightcolor="green")
image		กำหนดรูปภาพให้กับปุ่มแทนการใช้ข้อความ เช่น image = PhotoImage(file='printer.png') B = Button(root, text = "Hello", image=image)
justify		กำหนดตำแหน่งการแสดงผลข้อความบนปุ่ม โดย LEFT=ขวา ข้อความซิดด้านซ้ายของปุ่ม, RIGHT=ซิดด้านขวา, CENTER=กลาง ตรงกลางปุ่ม เช่น Button(root, text = "Hello\nPython\nLanguage", justify=LEFT)
padx		เติมข้อความว่าง (Padding) ด้านซ้ายและขวาของข้อความในปุ่ม เช่น Button(root, text = "Hello", padx=5)
pady		เติมข้อความว่าง (Padding) ด้านบนและล่างของข้อความในปุ่ม เช่น Button(root, text = "Hello", pady=5)
relief		กำหนดลักษณะของปุ่มในแบบ 3-D เช่น Button(root, text = "Hello", relief=GROOVE)
state		กำหนดให้ปุ่มทำงานหรือไม่ทำงาน ถ้ากำหนดเป็น DISABLED ปุ่มจะ ไม่สามารถกดได้ แต่ถ้ากำหนดเป็น ACTIVE จะสามารถกดปุ่มได้ เช่น Button(root, text = "Hello", state=DISABLED)

underline	 <p>ตัวอักษรของปุ่มจะถูกขีดเส้นใต้ โดยค่า -1 คือปุ่มจะไม่ถูกขีดเส้น แต่ตัวเลขอื่นๆ ที่เป็นค่าบวกจะทำให้ตัวอักษรบนปุ่มถูกขีดเส้น (0 คืออักษรตัวแรก) เช่น Button(root, text = "Hello", underline=1)</p>
width	 <p>กำหนดความกว้างของปุ่ม (กรณีปุ่มที่ถูกสร้างด้วยรูปภาพจะมีหน่วยเป็นพิกเซล) เช่น Button(root, text ="Hello", width=5)</p>
wraplength	 <p>เมื่อค่าดังกล่าวถูกกำหนดเป็นจำนวนเต็มบวก ข้อความบนปุ่มจะถูกจำกัดพื้นที่ โดยจะแสดงผลอยู่ในขอบเขตที่กำหนดใน wraplength เท่านั้น เช่น ข้อความ "Hello" จะใช้พื้นที่ในการแสดงผลเท่ากับ 28 พิกเซล เมื่อกำหนด wraplength=10 จะทำให้ข้อความแสดงในแนวตั้ง ตัวอย่าง Button(root, text = "Hello", wraplength=10)</p>

Widget ชนิดปุ่มมีเมธอดที่ช่วยเสริมในการทำงานของปุ่มคือ

- เมื่อ dot **flash()** ทำหน้าที่ว่าดปุ่มใหม่ เช่น
B = Button(root, text = "Hello")
B.**flash()**
 - เมื่อ dot **invoke()** บังคับให้คำสั่งอพชัน command ที่กำหนดไว้ในปุ่มทำงานทันที เช่น
B = Button(root, text = "Hello", **command=callBack**)
B.**invoke()**

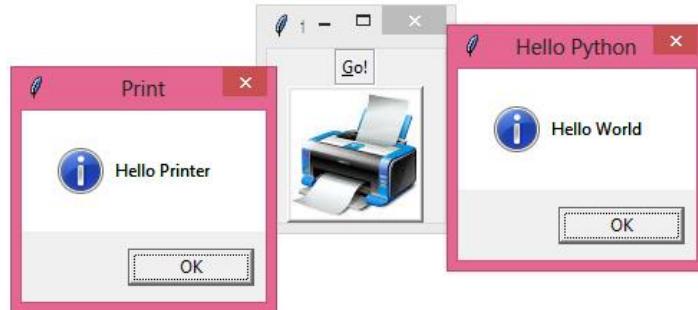
เมื่อสั่งรันโปรแกรมจะส่งผลให้ไฟ好不好เรียกเมื่อ dot callback() มาทำงานทันทีโดยไม่ต้องคลิกที่ปุ่ม

Program Example 19.12: Button

```
1 from tkinter import *
2 import tkinter
3
4 root = tkinter.Tk()
5
6 def helloCallBack():
7     messagebox.showinfo( "Hello Python", "Hello World")
8
9 def printCallBack():
10    messagebox.showinfo( "Print", "Hello Printer")
11
12 image = PhotoImage(file='printer.png')
13 B1 = Button(root, text ="Go!", relief=GROOVE, underline=0,
14 activebackground="yellow", activeforeground="red", command =
15 helloCallBack)
16 B2 = Button(root, image=image, padx=30, pady=20,
17 command=printCallBack)
18 B1.pack(expand=True)
19 B2.pack()
```

18 | root.mainloop()

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง ให้กดลงคlikที่ปุ่ม Go! และปุ่มเครื่องพิมพ์

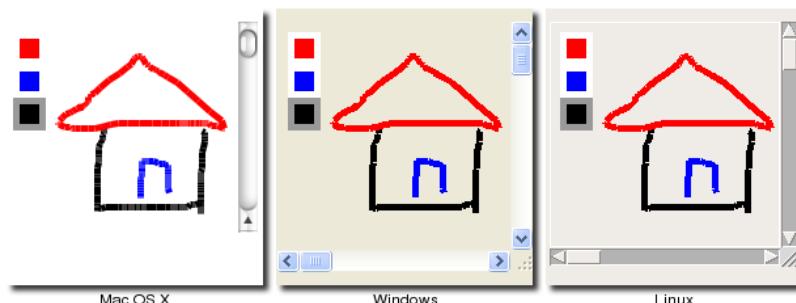


จากตัวอย่างโปรแกรมที่ 19.12 และการสร้างและใช้งาน Button (ปุ่ม) โดยบรรทัดที่ 6 โปรแกรมสร้างเมื่อ helloCallBack() มีหน้าที่สร้างกล่องข้อความโดยพิมพ์ข้อความว่า "Hello World" ออกจอภาพ บรรทัดที่ 9 โปรแกรมสร้างเมื่อ printCallBack() มีหน้าที่สร้างกล่องข้อความโดยพิมพ์ข้อความว่า "Hello Printer" ออกจอภาพ บรรทัดที่ 12 สร้างอ้อปเจ็กต์ image ที่เชื่อมโยงไปยังรูปภาพชื่อว่า "printer.png" เพื่อใช้สำหรับแสดงบนปุ่ม บรรทัดที่ 13 โปรแกรมสร้างปุ่มโดยมีข้อความบนปุ่มคือ "Go!" (text ="Go!"), ขีดเส้นใต้ที่ตัวอักษร "G" (underline=0), ขอบของปุ่มเป็นแบบร่องลึก (relief=GROOVE), เมื่อปุ่มถูกกด ปุ่มจะเป็นสีเหลือง (activebackground="yellow"), เมื่อปุ่มถูกกดข้อความจะเป็นสีแดง (activeforeground="red") และเมื่อปุ่มดังกล่าวถูกคลิก โปรแกรมจะเรียกใช้เมื่อ helloCallBack()

บรรทัดที่ 14 โปรแกรมสร้างปุ่มโดยมีภาพบนปุ่มคือ "printer.png" (image=image) เมื่อปุ่มดังกล่าวถูกคลิก โปรแกรมจะเรียกใช้เมื่อ printCallBack() ผลลัพธ์ที่ได้แสดงดังรูปด้านบน

3. Canvas (ผืนผ้าใบ)

Canvas คือ Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยม มีเป้าหมายเพื่อใช้สำหรับจัดวางรูปภาพ เฟรม ข้อความ หรือว่าตรุปภาพ ที่มีความซับซ้อนได้ รูปร่างของ Canvas แสดงในรูปที่ 19.15



รูปที่ 19.15 แสดง Canvas บนระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Canvas คือ

```
c = Canvas( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Canvas แสดงในตารางด้านล่าง

Option	คำอธิบาย
confine	กำหนดให้ Canvas สามารถเลื่อน Scroll ได้ (ดีฟอลต์เท่ากับ True) เช่น Canvas(root, bd=5, relief=GROOVE, height=250, width=300, confine=False)
cursor	กำหนดรูปแบบของเมาส์ เครื่องเมาส์จะเปลี่ยนรูปเมื่อเคลื่อนเมาส์ทับบน Canvas เช่น Canvas(root, bd=5, relief=GROOVE, height=250, width=300, cursor="hand1")
bd	 กำหนดขนาดความกว้างของ Canvas มีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Canvas(root, bd=3 , relief=GROOVE, height=250, width=300)
bg	 กำหนดสีพื้นหลังของ Canvas เช่น Canvas(root, bd=3, relief=GROOVE, bg="blue" , height=250, width=300)
scrollregion	กำหนดพื้นที่ที่ Canvas สามารถขยายได้สูงสุดเท่าใด โดยขอบเขตพื้นที่กำหนดในตัวแปรชนิดทัพเพิล ซึ่งมีรูปแบบ tuple(w, e, n, s) โดย w คือขอบด้านซ้าย, e คือขอบด้านขวา, n คือด้านบน และ s คือด้านล่าง เช่น Canvas(root, bd=5, relief=GROOVE, scrollregion=(0, 0, 500, 500))
xscrollincrement	กำหนดขนาดการเพิ่มขึ้นของจำนวนคอลัมน์ เมื่อ Canvas ใช้ Scrollbar ในแนวอน ใช้ในกรณีที่ Canvas ต้องการแสดงผลมากกว่าขอบเขตที่ Canvas กำหนดไว้ เช่น Canvas(frame, xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set, xscrollincrement=10 , yscrollincrement=10)
height	กำหนดความสูงของ Canvas เช่น Canvas(root, bd=3, relief=GROOVE, height=250 , width=300)
highlightcolor	กำหนดແຕບສีເມື່ອ Canvas ໄດ້ຮັບຄວາມສຳໃຈ (Focus) เช่น Canvas(root, bd=5, relief=GROOVE, height=250, width=300, highlightcolor="green")

xscrollcommand	กำหนดให้ Canvas สามารถใช้งาน Scrollbar ในแนวตั้งได้ เช่น Canvas(frame, xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set)
yscrollincrement	เหมือนกับ xscrollincrement แต่เปลี่ยนเป็นแนวตั้งแทน เช่น Canvas(frame, xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set, xscrollincrement=10, yscrollincrement=10)
yscrollcommand	กำหนดให้ Canvas สามารถใช้งาน Scrollbar ในแนวตั้งได้ เช่น Canvas(frame, xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set)
relief	กำหนดลักษณะขอบของ Canvas ในแบบ 3-D เช่น Canvas(root, bd=5, relief=GROOVE , height=250, width=300)
width	กำหนดความกว้างของ Canvas เช่น Canvas(root, bd=3, relief=GROOVE, height=250, width=300)

จากที่กล่าวมาแล้วว่า Canvas สามารถวาดรูปต่างๆ ลงบน Canvas ได้ ดังนั้นเมื่อต้องป้อนจึงใช้งานร่วมกับ Canvas ได้เป็นอย่างดี

- การวาดเส้นโค้งบน Canvas ด้วยเมธอด `create_arc()` ตัวอย่างเช่น

```
coord = 10, 50, 240, 210
arc = canvas.create_arc(coord, start=0, extent=150,
fill="blue")
```

- การสร้างรูปภาพบน Canvas ด้วยเมธอด `create_image()` ตัวอย่างเช่น

```
filename = PhotoImage(file = "sunshine.gif")
image = canvas.create_image(50, 50, anchor=NE,
image=filename)
```

- การวาดเส้นบน Canvas ด้วยเมธอด `create_line()` ตัวอย่างเช่น

```
line = canvas.create_line(x0, y0, x1, y1, ..., xn, yn,
options)
```

- การวาดรูปวงรีบน Canvas ด้วยเมธอด `create_oval()` ตัวอย่างเช่น

```
oval = canvas.create_oval(x0, y0, x1, y1, options)
```

- การวาดรูปหลายเหลี่ยมบน Canvas ด้วยเมธอด `create_polygon()` ตัวอย่างเช่น

```
polygon= canvas.create_polygon(x0, y0, x1, y1,...xn, yn,
options)
```

- การวาดรูปสี่เหลี่ยมบน Canvas ด้วยเมธอด `create_rectangle()` ตัวอย่างเช่น

```
rect = canvas.create_rectangle(50, 25, 150, 75,
fill="blue")
```

- การลบภาพวาดทั้งหมดออกจาก Canvas ด้วยเมธอด `create_delete()` ตัวอย่างเช่น

```
rect = canvas.delete(oval) #remove oval from Canvas
rect = canvas.delete(ALL) #remove ALL from Canvas
```

การสร้างรูปต่างๆ บน Canvas สามารถอ่านข้อมูลเพิ่มเติมได้ที่เว็บไซต์

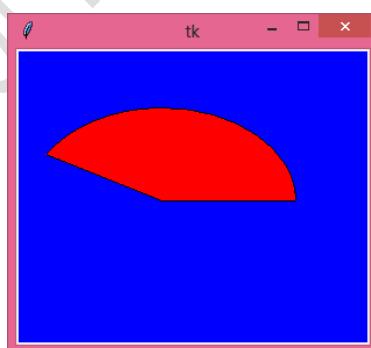
http://www.java2s.com/Tutorial/Python/0360__Tkinter/0100__Canvas.htm

สำหรับตัวอย่างการสร้างและใช้งาน Canvas แสดงในโปรแกรมตัวอย่างที่ 19.13 ดังนี้

Program Example 19.13: *Canvas*

```
1 import tkinter
2 from tkinter import *
3
4 root = tkinter.Tk()
5
6 C = tkinter.Canvas(root, bg="blue", height=250, width=300)
7
8 coord = 10, 50, 240, 210
9 arc = C.create_arc(coord, start=0, extent=150, fill="red")
10
11 C.pack()
12 root.mainloop()
```

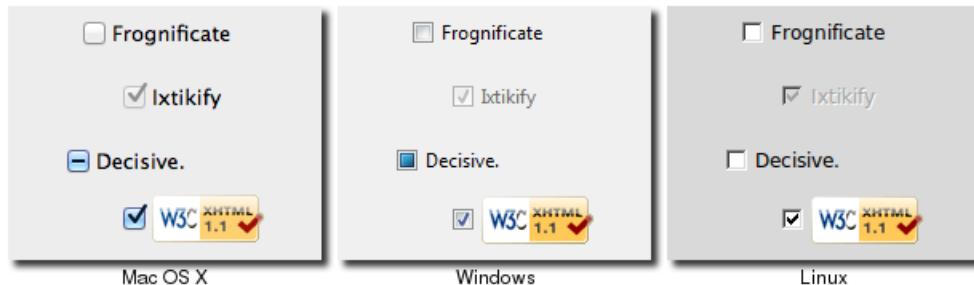
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.13 แสดงการสร้างและใช้งาน Canvas โดยบรรทัดที่ 6 โปรแกรมสร้าง Canvas ที่มีสีพื้นหลังเป็นสีน้ำเงิน (`bg="blue"`) มีความกว้างเท่ากับ 300 และสูงเท่ากับ 250 บรรทัดที่ 6 กำหนดตำแหน่งของ `coord` ซึ่งเป็นตัวแปรชนิดทัพเพิลเท่ากับ 10, 50, 240, 210 ตามลำดับ บรรทัดที่ 9 โปรแกรมทำการวาดวงกลมเสี้ยวสีแดงลงบน Canvas ดังรูปด้านบน

4. Checkbutton

Checkbutton คือ Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยมเล็กๆ เพื่อให้ผู้ใช้สามารถเลือกได้โดยการคลิกบน Checkbutton ดังกล่าว ผู้ใช้สามารถเลือกได้มากกว่า 1 ตัวเลือก เช่น การเลือกคำตอบที่มี 4 ตัวเลือก ก, ข, ค และ ง เป็นต้น รูปร่างของ Checkbutton แสดงในรูปที่ 19.16



รูปที่ 19.16 แสดงรูปร่างของ Checkbutton ในระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Checkbutton คือ

```
cb = Checkbutton( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Checkbutton แสดงในตารางด้านล่าง

Option	คำอธิบาย
activebackground	 กำหนดสีพื้นหลังของ Checkbutton เมื่อผู้ใช้คลิกบันปุ่ม เช่น Checkbutton(root, text = "Music", activebackground ="gray")
activeforeground	 กำหนดสีข้อความบน Checkbutton เมื่อผู้ใช้คลิกบันปุ่ม เช่น Checkbutton(root, text = "Music", activebackground="gray", activeforeground "white")
bd	 กำหนดขนาดความกว้างของ Checkbutton มีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Checkbutton(root, text = "Music", bg="yellow", bd=5 , relief=GROOVE)
bg	 กำหนดสีพื้นหลังของ Checkbutton เช่น Checkbutton(root, text = "Music", bg "yellow")
bitmap	แสดงภาพแบบ monochrome บน Checkbutton
command	ฟังชันหรือเมธอดที่จะถูกเรียกใช้เมื่อ Checkbutton ถูกกดหรือคลิก เช่น Checkbutton(root, text = "Music", command =myFunc)

fg	 Music	กำหนดสีของฟอนต์ เช่น Checkbutton(root, text = "Music", fg="red")
cursor	 Music	กำหนดรูปแบบของเมาส์ เชอร์เควอร์จะเปลี่ยนรูปเมื่อเคลื่อนเมาส์ทับบน Checkbutton เช่น Checkbutton(root, text = "Music", fg="red", cursor="hand1")
font	 Music	กำหนดรูปแบบของฟอนต์ของ Checkbutton เช่น Checkbutton(root, text = "Music", fg="red", font="Times 10 bold")
height	 Music	กำหนดความสูงของ Checkbutton เช่น Checkbutton(root, text = "Music", height=5, width=10, bg="pink")
width		กำหนดความกว้างของ Checkbutton เช่น Checkbutton(root, text = "Music", height=5, width=10, bg="pink")
highlightcolor		กำหนดແຄບສີເນື້ອ Checkbutton ໄດ້ຮັບຄວາມສຳໃຈ เช่น Checkbutton(root, text = "Music", height=5, width=10, highlightcolor="green")
image		กำหนดรูปภาพให้ກັບປຸ່ມແທນການໃຊ້ຂໍ້ຄວາມ เช่น image = PhotoImage(file='printer.png') Checkbutton(root, text = "Music", height=5, width=10, image=image)
justify	 Music Video	กำหนดตำแหน่งการแสดงผลข้อความນັ້ນ ໂດຍ LEFT=ວາງ ข້ອຄວາມຊິດດ້ານຫ້າຍຂອງນັ້ນ, RIGHT=ຊິດດ້ານຂວາ, CENTER=ວາງ ຕຽກລາງນັ້ນ เช่น Checkbutton(root, text = "MusicInVideo", justify=LEFT)
padx		ເຕີມຂໍ້ຄວາມວ່າງ (Padding) ດ້ານຫ້າຍແລະຂວາຂອງข້ອຄວາມໃນ Checkbutton เช่น Checkbutton(root, text = "Music", padx=5)
pady		ເຕີມຂໍ້ຄວາມວ່າງ (Padding) ດ້ານບັນແລະລ່າງຂອງข້ອຄວາມໃນ Checkbutton เช่น Checkbutton(root, text = "Music", pady=5)
relief	 Music	กำหนดລັກຜະນະຂອບຂອງ Checkbutton ໃນແບບ 3-D เช่น Checkbutton(root, text = "Music", relief=GROOVE)
state	 Music	กำหนดໃຫ້ Checkbutton ทำงานหรือໄມ່ทำงาน ຄໍາกำหนดເປັນ DISABLED Checkbutton ຈະໄມ່ทำงาน เช่น Checkbutton(root, text = "Music", state=DISABLED)
underline	 Music	ຕົວອັກຊ່າງຂອງ Checkbutton ຈະຖູກນື້ດເສັ້ນໄຕ (0 ຄືອັກຊ່າງຕົວແຮກ, -1 = ໄນນື້ດເສັ້ນໄຕ) เช่น Checkbutton(root, text = "Music", underline=1)
wraplength	 Music	ຂໍ້ຄວາມນັ້ນຈະຖູກຈຳກັດພື້ນທີ່ ໂດຍຈະແສດງຜລອຢູ່ໃນຂອບເຂດທີ່ กำหนดໃນ wraplength ເທົ່ານັ້ນ เช่น Checkbutton(root, text = "Music", wraplength=20)

onvalue	กำหนดค่าเริ่มต้นให้กับ Checkbutton เมื่อ Checkbutton ถูกคลิกเลือก โปรแกรมจะดึงค่าใน onvalue ไปใช้งาน เช่น Checkbutton(root, text = "Music", onvalue=1), หน้าที่ของ onvalue คือ จัดเตรียมค่าข้อมูลเพื่อให้เมธอดอื่นๆ นำไปใช้งานนั่นเอง (onvalue ในรูปแบบสตริงคือ "on")
offvalue	กำหนดค่าเริ่มต้นให้กับ Checkbutton เมื่อ Checkbutton ถูกคลิกยกเลิก โปรแกรมจะดึงค่าใน offvalue ไปใช้งาน เช่น Checkbutton(root, text = "Music", offvalue=0), หน้าที่ของ offvalue คือ จัดเตรียมค่าข้อมูลเพื่อให้เมธอดอื่นๆ นำไปใช้งานเช่นเดียวกับ onvalue (offvalue ในรูปแบบสตริงคือ "off")
selectcolor	 Music กำหนดสีของช่องว่างใน Checkbutton เช่น Checkbutton(root, text = "Music", selectcolor="red")
selectimage	กำหนดรูปภาพของช่องว่างใน checkbutton เช่น image = PhotoImage(file='printer.png') Checkbutton(root, text = "Music", selectimage=image)
text	กำหนดข้อความให้กับ Checkbutton ถ้าต้องการกำหนดข้อความมากกว่า 1 บรรทัดให้ใช้ \n เช่น "Music \n Audio \n Guitar"
variable	ใช้สำหรับดึงข้อมูลจาก Widgets หรือดึงข้อมูลจาก onvalue และ offvalue นั่นเอง โดยจะทำงานร่วมกับเมธอด IntVar() เมื่อข้อมูลใน onvalue/offvalue เป็นตัวเลข และทำงานร่วมกับเมธอด StringVar() เมื่อข้อมูลใน onvalue/offvalue เป็นสตริง เช่น Checkbutton(root, text = "Music", variable=IntVar(), onvalue=1, offvalue=0)

Widget ชนิด Checkbutton มีเมธอดที่ช่วยสนับสนุนการทำงานคือ

- เมธอด **deselect()** ทำหน้าที่เคลียร์ค่า Checkbutton ที่เลือกไว้ (turn-off) เช่น

```
C = Checkbutton(root, text = "Music")
C.deselect()
```
- เมธอด **flash()** ทำหน้าที่วัด Checkbutton ใหม่ เช่น

```
C = Checkbutton(root, text = "Music")
C.flash()
```
- เมธอด **invoke()** จะบังคับให้ทำการสั่นหลัง command ทันที เช่น

```
C = Checkbutton(root, text = "Music", command=callBack)
C.invoke()
```
- เมธอด **select()** เช็ต (turn-on) ค่าให้กับ Checkbutton เมื่อสถานะเดิมของ Checkbutton ไม่ถูกเลือกจะทำให้สถานะกลับเป็นถูกเลือกแทน เช่น

```
C = Checkbutton(root, text = "Music")
C.select()
```

- เมธอด `toggle()` สลับระหว่าง turn-on และ turn-off เมื่อสถานะเดิมของ Checkbutton ถูกกำหนดเป็น on เมื่อเรียกเมธอด `toggle()` จะทำให้ Checkbutton กลายเป็น off เช่น
`C = Checkbutton(root, text = "Music")`
`C.toggle()`

สำหรับตัวอย่างการสร้างและใช้งานปุ่ม แสดงในโปรแกรมตัวอย่างที่ 19.14 ดังนี้

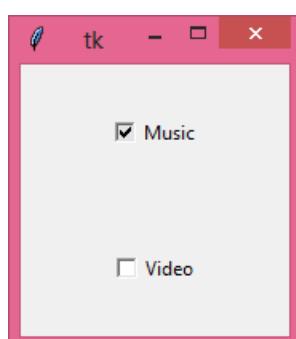
Program Example 19.14: Checkbutton

```

1   from tkinter import *
2   import tkinter
3
4   root = tkinter.Tk()
5   CheckVar1 = StringVar()
6   CheckVar2 = IntVar()
7
8   def checkCallBack():
9       C1.select()
10      C2.toggle()
11      print(CheckVar1.get())
12      print(CheckVar2.get())
13
14  C1 = Checkbutton(root, text = "Music", variable = CheckVar1,
15  onvalue = "on", offvalue = "off", height=5, width = 20,
16  command=checkCallBack)
17  C2 = Checkbutton(root, text = "Video", variable = CheckVar2,
18  onvalue = 1, offvalue = 0, height=5, width = 20,
19  command=checkCallBack)
20
21  C1.pack()
22  C2.pack()
23  root.mainloop()

```

ผลลัพธ์ที่ได้มีส่วนของโปรแกรมดังรูปด้านล่าง ให้ทดลองคลิกที่ Checkbutton และสังเกตการเปลี่ยนแปลง



จากตัวอย่างโปรแกรมที่ 19.14 แสดงการสร้างและใช้งาน Checkbutton บรรทัดที่ 5 เป็นการสร้างตัวแปรชื่อ `CheckVar1` เพื่อใช้สำหรับเก็บค่าข้อมูลชนิดสตริง ในที่นี่คือ 'on' หรือ 'off' ที่เกิดขึ้นจากการคลิกที่ Checkbutton (C1) โดยตัวแปรดังกล่าวสร้างขึ้นจากคลาสชื่อ `StringVar()` บรรทัดที่ 6 เป็นการสร้างตัวแปรชื่อ `CheckVar2` ซึ่งเป็นชนิดจำนวนเต็ม ในที่นี่คือ 0 หรือ 1 ที่เกิดจากการคลิก Checkbutton

(C2) โดยสร้างมาจากคลาสชื่อ IntVar() บรรทัดที่ 8 โปรแกรมสร้างฟังชันชื่อ checkCallBack() เพื่อทดสอบการทำงานของ Checkbutton C1 และ C2 โดยฟังชันดังกล่าวเรียกใช้งานเมื่อ select() และ toggle() พร้อมกับพิมพ์ค่า CheckVar1 และ CheckVar2 ออกทางจอภาพเมื่อผู้ใช้มีการคลิกที่ Checkbutton

บรรทัดที่ 14 สร้าง Widget Checkbutton ชื่อ C1 ที่มีข้อความว่า 'Music' มีความสูงเท่ากับ 5 ความกว้างเท่ากับ 20 เมื่อ Checkbutton C1 ถูกคลิกเลือกจะนำค่าใน onvalue ('on') เก็บไว้ในตัวแปร CheckVar1 ทันที แต่ถ้า C1 ไม่ถูกเลือก โปรแกรมจะนำค่า offvalue ('off') เก็บไว้ในตัวแปร CheckVar1 แทน ใน Checkbutton C1 โปรแกรมผังคำสั่งเอาไว้ ถ้ามีการคลิกที่ Checkbutton C1 โปรแกรมจะเรียกใช้เมธอด checkCallBack() ทันที (command=checkCallBack)

บรรทัดที่ 15 สร้าง Widget Checkbutton ชื่อ C2 ที่มีข้อความว่า 'Video' มีความสูงเท่ากับ 5 ความกว้างเท่ากับ 20 เมื่อ Checkbutton C2 ถูกคลิกเลือกจะนำค่าใน onvalue (1) เก็บไว้ในตัวแปร CheckVar2 ทันที แต่ถ้า C2 ไม่ถูกเลือก โปรแกรมจะนำค่า offvalue (0) เก็บไว้ในตัวแปร CheckVar2 แทน ใน Checkbutton C2 โปรแกรมผังคำสั่งเอาไว้ ถ้ามีการคลิกที่ Checkbutton C2 โปรแกรมจะเรียกใช้เมธอด checkCallBack() ผลลัพธ์ที่ได้แสดงดังในรูปด้านบน

5. Entry (นำเข้าข้อมูล)

Entry คือ Widget ที่มีลักษณะเป็นกล่องข้อความ เพื่อใช้รับข้อมูลจากผู้ใช้เข้ามา ประมวลผลในโปรแกรม เช่น การป้อนชื่อ-สกุล รหัสผ่าน เป็นต้น รูปร่างของ Entry แสดงในรูปที่ 19.17



รูปที่ 19.17 แสดงรูปแบบของ Entry บนระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Entry คือ

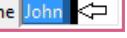
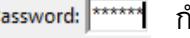
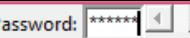
```
e = Entry( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)

○ option คือ คุณสมบัติต่างๆ ของ Entry

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, highlightcolor, justify, relief และ state สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
exportselection	โดยปกติเมื่อผู้ใช้เลือกข้อความภายใน Entry ข้อมูลที่ถูกเลือกจะถูกส่งไปเก็บไว้ในคลิปบอร์ด (Clipboard) โดยอัตโนมัติ เมื่อไม่ต้องการให้ข้อมูลดังกล่าวถูกส่งไปยังคลิปบอร์ด ให้กำหนด exportselection = 0 เช่น Entry(root, bd=5, exportselection=0)
selectbackground	User Name  กำหนดสีพื้นหลังของข้อความเมื่อข้อความดังกล่าวถูกเลือก (highlight) เช่น Entry(root, bd=5, selectbackground="red")
selectborderwidth	User Name  กำหนดขนาดความกว้างของขอบรอบๆ ข้อความที่ถูกเลือก (ค่าเดียวกันทั้งสองข้าง) เช่น Entry(root, bd=3, width=10, selectborderwidth=10)
selectforeground	User Name  กำหนดสีของข้อความใน Entry เมื่อข้อความดังกล่าวถูกเลือก เช่น Entry(root, bd=3, width=10, selectforeground="red")
show	Password:  ข้อความที่ป้อนเข้าไปใน Entry จะมีลักษณะเป็น Clear text (ไม่มีการเข้ารหัส) แต่บางครั้งผู้ใช้งานจำเป็นต้องซ่อนข้อความดังกล่าว เช่น รหัสผ่าน เป็นต้น สามารถกำหนดโดย show="*" เช่น Entry(root, bd=3, width=10, show="*")
textvariable	Password:  กำหนดตัวแปรสำหรับใช้เก็บค่าที่เกิดขึ้นจากการป้อนข้อมูลลงใน Entry โดยทำงานร่วมกับคลาส StringVar() เช่น entryVar = StringVar() Entry(root, bd=3, width=10, show="*", textvariable=entryVar)
width	Password:  กำหนดขนาดความกว้างของ Entry ที่แสดงผล เช่น Entry(root, bd=3, width=5)
xscrollcommand	Password:  เมื่อคาดว่าผู้ใช้งานจะป้อนข้อมูลเกินความกว้าง (Width) ของ Entry ที่กำหนดไว้ สามารถใช้ xscrollcommand (แท็บสไลด์ในแนวนอน) ทำงานร่วมกับ Entry ได้ เช่น scrollbar = Scrollbar(root, orient=HORIZONTAL) Entry(root, bd=3, width=5, show="*", xscrollcommand=scrollbar.set)

Widget ชนิด Entry มีเมธอดที่ช่วยสนับสนุนการทำงานคือ

- เมธอด `delete(first, last=None)` ทำหน้าที่ลบตัวอักษรออกจาก Entry โดยกำหนดตำแหน่งตัวอักษรเริ่มต้นที่พารามิเตอร์ `first` และตำแหน่งตัวอักษรตัวสุดท้ายในพารามิเตอร์ `last` ถ้าไม่ได้กำหนดค่าให้กับพารามิเตอร์ `last` ตัวอักษรตัวแรกจะถูกลบเพียงตัวเดียวเท่านั้น เช่น

```
E = Entry(root, bd=3, width=5, show="*")
E.delete(0, END)
```
- เมธอด `get()` ทำหน้าดึงข้อมูลจาก Entry เป็นสตริง เช่น

```
E = Entry(root, bd=3, width=5, show="*")
E.get()
```
- เมธอด `icursor(index)` กำหนดตำแหน่งของเคอร์เซอร์ในข้อความของ Entry ผ่านพารามิเตอร์ `index` เช่น

```
E = Entry(root, bd=3, width=5)
E.icursor(3)
```
- เมธอด `index(index)` เลื่อนตัวชี้ไปยังตำแหน่งที่ต้องการของข้อความใน Entry เช่น

```
E = Entry(root, bd=3, width=5)
E.index(3)
```
- เมธอด `insert(index, s)` เพิ่มสตริงในตำแหน่ง (`index`) ที่กำหนด เช่น

```
E = Entry(root, bd=3, width=5)
E.insert(3, "Hello")
```
- เมธอด `select_adjust(index)` เมธอดนี้ถูกใช้เพื่อยืนยันว่าตัวอักษรที่เลือกตรงตามที่ระบุไว้จริงๆ เช่น

```
E = Entry(root, bd=3, width=5)
E.select_adjust(3)
```
- เมธอด `select_clear()` เคลียร์แทบ `highlight` บนข้อความที่ถูกเลือกใน Entry เช่น

```
E = Entry(root, bd=3, width=5)
E.select_clear()
```
- เมธอด `select_from(index)` กำหนดตำแหน่ง ANCHOR ด้วย `index` เช่น

```
E = Entry(root, bd=3, width=5)
E.select_from(3)
```
- เมธอด `select_present()` เมื่อข้อความถูกเลือก (`highlight`) จะคืนค่าเป็น `True` แต่ไม่ถูกเลือกจะคืนค่าเป็น `False` เช่น

```
E = Entry(root, bd=3, width=5)
E.select_present()
```
- เมธอด `select_range(start, end)` เลือกช่วงของข้อความที่ต้องการ โดยกำหนดตำแหน่งเริ่มต้น (`start`) และตำแหน่งสิ้นสุด (`end`) เช่น

```
E = Entry(root, bd=3, width=5)
E.select_range(2, 4)
```
- เมธอด `select_to(index)` เลือกช่วงของข้อความตั้งแต่ ANCHOR ไปถึงตัวอักษรตัวสุดท้าย เช่น

```
E = Entry(root, bd=3, width=5)
E.select_to(5)
```

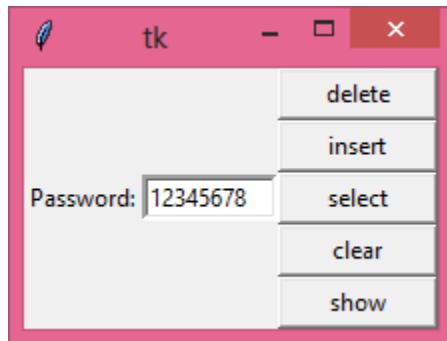
สำหรับตัวอย่างการสร้างและใช้งาน Entry แสดงในโปรแกรมตัวอย่างที่ 19.15 ดังนี้

Program Example 19.15: Entry

```

1   from tkinter import *
2
3   root = Tk()
4   entryVar = StringVar()
5
6   def deleteCallBack():
7       E1.delete(2, 4)
8
9   def insertCallBack():
10      E1.insert(3, "Hello")
11
12  def selectCallBack():
13      E1.select_range(2, 4)
14
15  def clearCallBack():
16      E1.select_clear()
17
18  def showCallBack():
19      print(entryVar.get())
20
21  L1 = Label(root, text="Password:")
22  L1.pack(side = LEFT)
23  E1 = Entry(root, bd=3, width=10, textvariable=entryVar)
24  E1.pack(side = LEFT)
25  B1 = Button(root, text="delete", width=10,
26             command=deleteCallBack)
27  B1.pack()
28  B2 = Button(root, text="insert", width=10,
29             command=insertCallBack)
30  B2.pack()
31  B3 = Button(root, text="select", width=10,
32             command=selectCallBack)
33  B3.pack()
34  B4 = Button(root, text="clear", width=10,
35             command=clearCallBack)
36  B4.pack()
37
38  B5 = Button(root, text="show", width=10,
39             command=showCallBack)
40  B5.pack()
41
42  root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง ให้กดลงปุ่มข้อมูลและคลิกเลือกปุ่มต่างๆ พร้อมกับสังเกตการเปลี่ยนแปลง

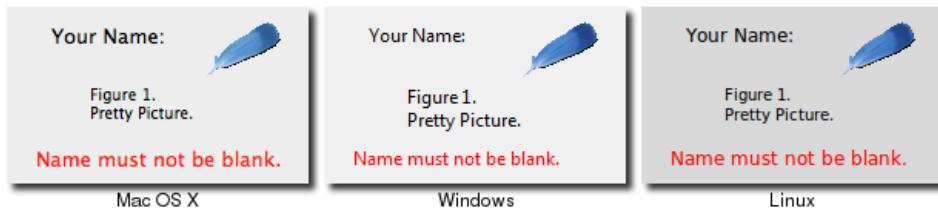


จากตัวอย่างโปรแกรมที่ 19.15 แสดงการสร้างและใช้งาน Entry บรรทัดที่ 4 สร้างตัวแปรชื่อ entryVar เพื่อใช้สำหรับเก็บค่าข้อมูลชนิดสตริงที่ป้อนลงใน Entry โดยตัวแปรดังกล่าวสร้างขึ้นจากคลาสชื่อ StringVar() บรรทัดที่ 6 สร้างฟังชันชื่อ deleteCallBack() ทำหน้าที่ลบข้อความใน Entry โดยการระบุตำแหน่งเริ่มต้น (first) และสิ้นสุด (last) ของข้อความด้วยเมธอด delete(first, last) บรรทัดที่ 9 สร้างฟังชันชื่อ insertCallBack() ทำหน้าที่แทรกหรือเพิ่มข้อความลงใน Entry ด้วยเมธอด insert(index, s) โดย index คือตำแหน่งที่ต้องการเพิ่มข้อความลงใน Entry และ s คือข้อความ บรรทัดที่ 12 สร้างฟังชัน selectCallBack() ทำหน้าที่เลือกช่วงของข้อความ (highlight) ใน Entry โดยใช้เมธอด selectRange(start, end) บรรทัดที่ 18 สร้างฟังชันชื่อ getCallBack() ทำหน้าที่แสดงผลข้อมูลที่อยู่ใน Entry ผ่านตัวแปรชนิดสตริงชื่อ entryVar โดยใช้เมธอด get()

บรรทัดที่ 21 โปรแกรมสร้าง Label เพื่อพิมพ์คำว่า 'Password' ออกจอภาพ บรรทัดที่ 23 สร้าง Entry ที่มีความกว้างของเท่ากับ 10 เมื่อผู้ใช้ป้อนข้อมูลใดๆ ลงใน Entry ข้อความเหล่าจะถูกเก็บไว้ในตัวแปร entryVar (textvariable=entryVar) บรรทัดที่ 25 สร้างปุ่มมีข้อความบนปุ่มคือ 'delete' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน deleteCallBack() เข้ามาทำงานทันที บรรทัดที่ 27 สร้างปุ่มมีข้อความคือ 'insert' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน insertCallBack() บรรทัดที่ 29 สร้างปุ่มมีข้อความคือ 'select' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน selectCallBack() บรรทัดที่ 31 สร้างปุ่มมีข้อความคือ 'clear' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน clearCallBack() บรรทัดที่ 33 สร้างปุ่มมีข้อความคือ 'show' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน showCallBack() ผลลัพธ์แสดงดังรูปด้านบน

6. Label (เลเบลหรือป้ายชื่อ)

Label คือ Widget ที่มีลักษณะเป็นป้ายของข้อความ เพื่อใช้แสดงข้อความต่างๆ ผู้ใช้งานทราบ เช่น ป้ายชื่อผู้ใช้งาน (User label) ป้ายชื่อรหัสผ่าน (Password label) เป็นต้น รูปร่างของ Label แสดงในรูปที่ 19.18



รูปที่ 19.18 แสดงรูปแบบของเลเบลนระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Label คือ

```
l = Label( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Label

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, height, width, image, tjustify, padx, pady, relief, underline และ wraplength สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
anchor	User Name: กำหนดตำแหน่งการวาง Label โดยการกำหนดทิศคือ N, W, S, E เป็นต้น (อ่านข้อมูลเพิ่มในหัวข้อ คุณสมบัติพื้นฐานของ Widgets) ค่าดีฟอลต์คือ กลางวินโดว์ เช่น Label(root, anchor=NW, text="User Name:")
text	User Name: กำหนดข้อความให้แสดงบน Label เช่น Label(root, text='User Name:', font=("Helvetica", 9), fg="red")
bitmap	🚫 กำหนดรูปภาพบิตแม็บที่ต้องการแสดงบน Label เช่น Label(root, bitmap="error", fg="red")
textvariable	กำหนดตัวแปรสำหรับใช้เก็บข้อความของ Label โดยทำงานร่วมกับคลาส StringVar() เช่น labelVar = StringVar() Label(root, textvariable=labelVar, text='User Name:')

คลาส StringVar() และคลาส IntVar() มีเมธอดสำคัญที่ช่วยสนับสนุนการทำงานของ Widgets คือ

- เมธอด get() ทำหน้าที่ดึงข้อมูลตากตัวแปรคลาส StringVar() และ IntVar() มาแสดงผล เช่น


```
stringVariable = StringVar()
Label(root, text="User Name:", textvariable=stringVariable)
print(stringVariable.get()) # result = "User Name:"
```

- เมธอด `set()` ทำหน้าที่กำหนดข้อความใหม่ให้กับ Label เช่น

```
stringVariable = StringVar()
Label(root, text="User Name:", textvariable=stringVariable)
print(stringVariable.set("User:")) # result = "User"
```

สำหรับตัวอย่างการสร้างและใช้งาน Label แสดงในโปรแกรมตัวอย่างที่ 19.16 ดังนี้

Program Example 19.16: Label

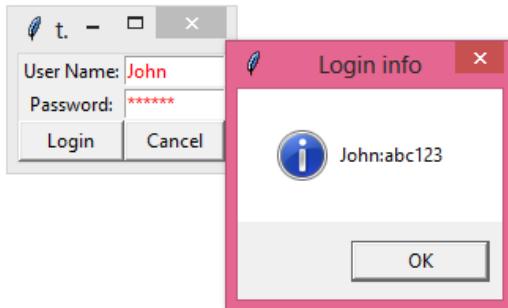
```
1   from tkinter import *
2   root = Tk()
3
4   userVar = StringVar()
5   passwdVar = StringVar()
6
7   def showLoginInfo():
8       msg = userVar.get() + ":" + passwdVar.get()
9       print(msg)
10      messagebox.showinfo("Login info", msg)
11
12  def clearLogin():
13      userVar.set("")
14      passwdVar.set("")
15
16  L1 = Label(root, text='User Name:')
17  L1.grid(column=0, row=0, sticky=(W, E))
18  E1 = Entry(root, width=10, fg="red", textvariable=userVar)
19  E1.grid(column=1, row=0, sticky=(W, E))
20  L2 = Label(root, text='Password:')
21  L2.grid(column=0, row=1, sticky=(W, E))
22  E2 = Entry(root, width=10, show="*", fg="red",
23             textvariable=passwdVar)
24  E2.grid(column=1, row=1, sticky=(W, E))
25  B1 = Button(root, text='Login', command=showLoginInfo)
26  B1.grid(column=0, row=2, sticky=(W, E))
27  B2 = Button(root, text='Cancel', command=clearLogin)
28  B2.grid(column=1, row=2, sticky=(W, E))
29
30  root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง ให้กดลงปุ่นชื่อช่องและคลิกเลือกปุ่ม Login และ Cancel พร้อมกับสังเกตการเปลี่ยนแปลง

ตำแหน่งการวาง Widgets ต่างๆ ลงบน grid(column, row) และแสดงได้ดังนี้

คอลัมน์/แก้ว	คอลัมน์ 0	คอลัมน์ 1
แก้ว 0	Widget L1 (User Name:)	Widget E1 (ช่องรับชื่อผู้ใช้)
แก้ว 1	Widget L2 (Password:)	Widget E2 (ช่องรับรหัสผ่าน)

ແກ້ວ 2	Widget B1 (ປຸ່ມ Login)	Widget B2 (ປຸ່ມ Cancel)
--------	------------------------	-------------------------

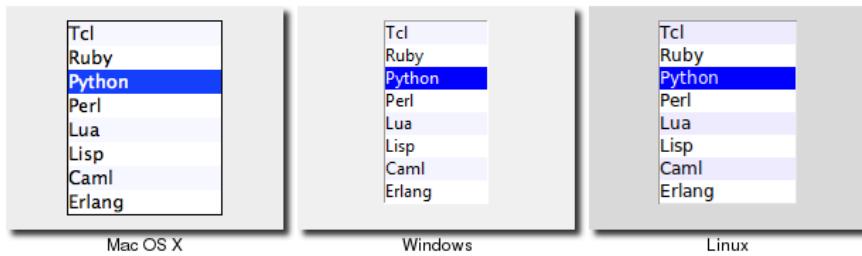


ຈາກຕ້ວອຍ່າງໂປຣແກຣມທີ 19.16 ແສດງການສ້າງແລະໃຊ້ງານ Label ບຣທັດທີ 4 ແລະ 5 ສ້າງຕໍ່ວິເພີ່ມ
userVar ແລະ passwordVar ເພື່ອໃຊ້ສໍາຮັບເກີບຄ່າຂໍ້ມູນນິດສຕຽງທີ່ປ້ອນລົງໃນ Entry ຂອງ User Name
ແລະ Password ຕາມລຳດັບ ບຣທັດທີ 7 ສ້າງພັ້ນທີ່ showLoginInfo() ທ່ານ້າທີ່ແສດງຊື່ຜູ້ໃຊ້ງານແລະ
ຮັສຜ່ານດ້ວຍກລ່ອງຂໍ້ວິເພີ່ມຂໍ້ "Login info" ແລະ ພິມພົມຂໍ້ມູນອອກທາງ Python shell ດ້ວຍ ບຣທັດທີ 12
ສ້າງພັ້ນທີ່ clearLogin() ທ່ານ້າທີ່ລົບຊື່ຜູ້ໃຊ້ແລະ ຮັສຜ່ານອອກຈາກ Entry

ບຣທັດທີ 16 ໂປຣແກຣມສ້າງ Label ເພື່ອພິມພົມຄໍາວ່າ 'User Name:' ອອກຈອກພາບ ບຣທັດທີ 18
ສ້າງ Entry ທີ່ມີຄວາມກວ້າງຂອງເທົກກັບ 10, ຕົວອັກຊ່າທີ່ປ້ອນຈະຖຸກກຳຫັດໃຫ້ເປັນສືແດງ ແລະ ຂໍ້ມູນທີ່ປ້ອນຈະ
ເກີບໄວ້ໃນຕໍ່ວິເພີ່ມຂໍ້ userVar (textvariable=userVar) ບຣທັດທີ 20 ສ້າງ Label ເພື່ອພິມພົມຄໍາວ່າ
'Password:' ອອກຈອກພາບ ບຣທັດທີ 22 ສ້າງ Entry ທີ່ມີຄວາມກວ້າງຂອງເທົກກັບ 10 ຂໍ້ມູນທີ່ປ້ອນຈະຖຸກ
ເຂົ້າຮັສດ້ວຍຕົວອັກຊ່າ '*' ແລະ ເກີບຂໍ້ມູນໄວ້ໃນຕໍ່ວິເພີ່ມຂໍ້ passwordVar ບຣທັດທີ 25 ສ້າງປຸ່ມທີ່ມີຂໍ້ວິເພີ່ມ
ບໍ່ປຸ່ມຄື່ອ 'Login' ເມື່ອປຸ່ມດັ່ງກ່າວລ່າວຖຸກກົດ ໂປຣແກຣມຈະເຮັດວຽກພັ້ນທີ່ showLoginInfo() ເຂົ້າມາທຳນານທັນທີ
ບຣທັດທີ 27 ສ້າງປຸ່ມມີຂໍ້ວິເພີ່ມຄື່ອ 'Cancel' ເມື່ອປຸ່ມດັ່ງກ່າວລ່າວຖຸກກົດ ໂປຣແກຣມຈະເຮັດວຽກພັ້ນ
clearLogin() ຜລລັບຮັບແສດງດັ່ງຮູບດ້ານນີ້

7. Listbox

Listbox ຄື່ອ Widget ທີ່ມີລັກຊະນະເປັນຮາຍການຂອງສາມາຊີກທີ່ສ່ວນໃຫຍ່ມີຕໍ່ວາເລືອກມາກກວ່າ 1
ຕໍ່ວາເລືອກ ໂດຍຜູ້ໃຊ້ງານສາມາດເລືອກສາມາຊີກຈາກຮາຍການທີ່ຕັ້ງກ່າວໄດ້ເພີ່ມຕໍ່ວາເດືອນນີ້ ເຊັ່ນ
ປະເທດທີ່ອາສີຍອຸ່ງ ຄໍານໍາຫັນທີ່ ເປັນຕົ້ນ ຮູ່ປ່າງຂອງ Listbox ແສດງໃນຮູ່ທີ່ 19.19



รูปที่ 19.19 แสดงรูปแบบของ Listbox ในระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Listbox คือ

```
l = Listbox( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Listbox

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, highlightcolor, highlightthickness, justify, relief สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
selectbackground	กำหนดสีข้อความใน Listbox เมื่อสมาชิกถูกเลือก เช่น Listbox(root, selectbackground="red")
selectmode	กำหนดคุณสมบัติว่าจะเลือกสมาชิกจาก Listbox อย่างไร โดยมีโหมด ให้เลือกดังนี้คือ <ul style="list-style-type: none"> ● BROWSE: เลือกสมาชิกได้เพียงตัวเดียว เมื่อลากเมาส์ (drag) แท็บสี (highlight) จะวิ่งไปพร้อมๆ กับเมาส์ ● SINGLE: เลือกสมาชิกได้เพียงตัวเดียว และไม่สามารถลากเมาส์ได้ ● MULTIPLE: เลือกสมาชิกได้มากกว่า 1 ตัว โดยการคลิกเลือกสมาชิก แต่ละตัว ไม่สามารถลากเมาส์เพื่อเลือกสมาชิกหลายๆ ตัว พร้อมกันได้ ● EXTENDED: เลือกสมาชิกได้มากกว่า 1 ตัว โดยการลากเมาส์เพื่อเลือกสมาชิกหลายๆ ตัว พร้อมกันได้ เช่น Listbox(root, selectmode=EXTENDED)
xscrollcommand	สร้างแท็บสไลด์แนวนอน (HORIZONTAL) ให้กับ Listbox เช่น

	<pre>xscrollbar = Scrollbar(root, orient=VERTICAL) xscrollbar.pack(side=BOTTOM, fill=X) Listbox(root, selectmode=EXTENDED, xscrollcommand=xscrollbar.set)</pre>
yscrollcommand	<p>Python Perl C PHP</p>  <p>สร้างแท็บสไลด์แนวตั้งให้กับ Listbox (VERTICAL) เช่น</p> <pre>yscrollbar = Scrollbar(root, orient=VERTICAL) yscrollbar.pack(side=RIGHT, fill=Y) Listbox(root, selectmode=EXTENDED, yscrollcommand=yscrollbar.set)</pre>

Widget ชนิด Listbox มีเมธอดที่ช่วยสนับสนุนการทำงานคือ

- เมธอด `selection_set(first, last)` ทำหน้าที่เลือกช่วงสมาชิกที่ต้องการ โดยใช้กำหนดตำแหน่งสมาชิกตัวแรกในพารามิเตอร์ `first` และสมาชิกตัวสุดท้ายคือ `last` ถ้าระบุเฉพาะ `first` แสดงว่าเลือกสมาชิกจาก Listbox เพียงตัวเดียวเท่านั้น เช่น

```
Lbl = Listbox(root)
Lbl.insert(1, "Python")
Lbl.insert(2, "Perl")
Lbl.pack()
Lbl.selection_set(1) #Select only member from Listbox
Lbl.selection_set(0, END) #Select all member
```

- เมธอด `curselection()` ทำหน้าที่ดึงข้อมูลสมาชิกที่ถูกเลือกใน Listbox ค่าที่ส่งกลับจะเป็นข้อมูลชนิดทัพเพิล แต่ถ้าสมาชิกใน Listbox ไม่ได้ถูกเลือกไว้จะส่งค่ากลับเป็นทัพเพิลที่ว่างเปล่า เช่น

```
Lbl = Listbox(root)
Lbl.insert(1, "Python")
Lbl.insert(2, "Perl")
Lbl.insert(3, "C")
Lbl.pack()
Lbl.selection_set(0, 1)
print(Lbl.curselection())
ผลลัพธ์คือ (0, 1)
```

- เมธอด `delete(first, last=None)` ลบสมาชิกออกจาก Listbox ถ้ากำหนดตำแหน่ง `first` และ `last` แสดงว่าเป็นการลบสมาชิกแบบช่วง แต่ถ้ากำหนดเฉพาะ `first` จะเป็นการลบสมาชิกจาก Listbox เพียงตัวเดียวเท่านั้น เช่น

```
Lbl = Listbox(root)
Lbl.insert(1, "Python")
Lbl.insert(2, "Perl")
Lbl.insert(3, "C")
Lbl.pack()
Lbl.delete(1, 2)
```

- เมธอด `get(first, last=None)` ดึงสมาชิกจาก Listbox ถ้ากำหนดตำแหน่ง `first` และ `last` แสดงว่า เป็นการดึงสมาชิกแบบช่วง แต่ถ้ากำหนดเฉพาะ `first` จะเป็นการดึงสมาชิกจาก Listbox เพียง ตัวเดียวเท่านั้น ค่าที่ส่งคืนจากการเมธอดดังกล่าวเป็นชนิดทัพเพิล เช่น

```
Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.pack()
Lb1.get(1, 2)
ผลลัพธ์ที่ได้คือ ('Perl', 'C')
```

- เมธอด `index(i)` กำหนดตำแหน่งตัวซึ่ด้วย `index` เช่น
`Lb1.index(1)`
- เมธอด `insert(index, *elements)` เพิ่มสมาชิกตั้งแต่ 1 – n ตัว ลงใน Listbox โดยสามารถ กำหนดตำแหน่งที่จะเพิ่มด้วย `index` ถ้าต้องการเพิ่มสมาชิกในตำแหน่งท้ายของ Listbox ให้ทำ การกำหนด `index` เป็น END เช่น

```
Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.pack()
elements = ("C++", "Prolog")
for item in elements:
    Lb1.insert(END, item)
```

- เมธอด `nearest(y)` ดึงค่าตำแหน่งบนสุดของ Listbox เช่น
`Lb1.nearest(1)`
- เมธอด `see(index)` เปลี่ยนตำแหน่งของ Listbox โดยอ้างอิงจาก `index` เช่น
`Lb1.see(2)`
- เมธอด `size()` คำนวณจำนวนสมาชิกที่มีทั้งหมดใน Listbox เช่น
`Lb1.size()`

สำหรับตัวอย่างการสร้างและใช้งาน Listbox และแสดงในโปรแกรมตัวอย่างที่ 19.17, 19.18 ดังนี้

Program Example 19.17: Listbox

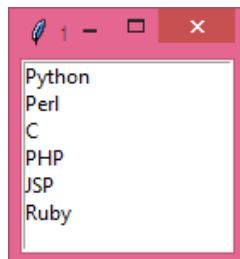
```
1 from tkinter import *
2 import tkinter
3
4 root = Tk()
5
⇒6 Lb1 = Listbox(root)
⇒7 Lb1.insert(1, "Python")
⇒8 Lb1.insert(2, "Perl")
⇒9 Lb1.insert(3, "C")
⇒10 Lb1.insert(4, "PHP")
```

```

⇒11 Lb1.insert(5, "JSP")
⇒12 Lb1.insert(6, "Ruby")
13
14 Lb1.pack()
15 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



ตัวอย่างโปรแกรมที่ 19.17 แสดงการสร้างและใช้งาน Listbox บรรทัดที่ 6 สร้าง Listbox ในหน้าต่างหลัก บรรทัดที่ 7 – 12 เป็นการเพิ่มสมาชิกให้กับ Listbox คือ 'Python', 'Perl', 'C', 'PHP', 'JSP' และ 'Ruby' ผลลัพธ์ที่ได้แสดงดังรูปด้านบน

ตัวอย่างโปรแกรมที่ 19.18 เป็นการประยุกต์ Listbox เข้ากับ scrollbar และ button ดังนี้

Program Example 19.18: Applied Listbox, Scrollbar, Button

```

1 from tkinter import *
2 import tkinter
3
4 root = Tk()
5
⇒6 def insertMember():
6     elements = ("C++", "Prolog")
7     for item in elements:
8         Lb1.insert(END, item)
9
10
⇒11 def deleteMember():
11     Lb1.delete(0, END)
12
13
⇒14 xscrollbar = Scrollbar(root, orient=HORIZONTAL)
⇒15 yscrollbar = Scrollbar(root, orient=VERTICAL)
16 xscrollbar.pack(side=BOTTOM, fill=X)
17 yscrollbar.pack(side=RIGHT, fill=Y)
18 Lb1 = Listbox(root, selectmode=EXTENDED,
19 xscrollcommand=xscrollbar.set)
20 Lb1.insert(1, "Python")
21 Lb1.insert(2, "Perl")
22 Lb1.insert(3, "C")
23 Lb1.insert(4, "PHP")
24 Lb1.insert(5, "JSP")
25 Lb1.insert(6, "Ruby")
26 Lb1.insert(7, "ASP")
27 Lb1.insert(8, "JAVA")
28 Lb1.insert(9, "HTML")
29 Lb1.pack()

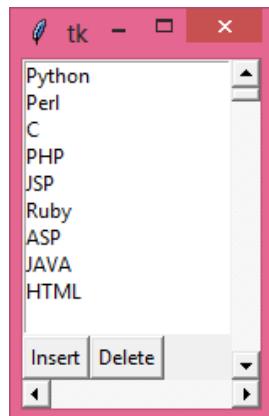
```

```

⇒29 B1 = Button(root, text="Insert", command=insertMember)
⇒30 B2 = Button(root, text="Delete", command=deleteMember)
31 B1.pack(side=LEFT)
32 B2.pack(side=LEFT)
33
34 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง ให้กดลงคุณลิกเลือกปุ่มต่างๆ พร้อมกับสังเกตการเปลี่ยนแปลง



จากตัวอย่างโปรแกรมที่ 19.18 บรรทัดที่ 6 สร้างเมธอดชื่อ `insertMember()` เพื่อทำหน้าที่เพิ่มสมาชิกให้กับ Listbox ทีละค่าตามลำดับ (เพิ่มข้อความว่า "C++" และ "Prolog") โดยเพิ่มต่อท้ายรายการของ Listbox บรรทัดที่ 11 สร้างฟังชันชื่อ `deleteMember()` ทำหน้าที่ลบสมาชิกทั้งหมดใน Listbox (`delete(0, END)`) บรรทัดที่ 14 สร้างแท็บสไลด์ในแนวนอน (HORIZONTAL Scrollbar) ชื่อว่า `xscrollbar` บรรทัดที่ 15 สร้างแท็บสไลด์ในแนวตั้ง (VERTICAL) ชื่อว่า `yscrollbar`

บรรทัดที่ 18 โปรแกรมสร้าง Listbox ชื่อว่า `Lb1` โดยผู้ใช้สามารถเลือกสมาชิกจาก Listbox ได้มากกว่า 1 รายการ (`selectmode=EXTENDED`) และ Listbox ดังกล่าวมีแท็บสไลด์อยู่ทางด้านขวาของ Listbox จากนั้นบรรทัดที่ 19 – 27 ทำการเพิ่มสมาชิกเข้าไปใน Listbox ทีละรายการ โดยเริ่มจาก "Python" และไปสิ้นสุดที่ "HTML" บรรทัดที่ 29 สร้างปุ่มชื่อ `B1` มีข้อความว่า "Insert" เมื่อผู้ใช้คลิกปุ่ม ดังกล่าวโปรแกรมจะเรียกใช้ฟังชัน `insertMember()` เข้ามาทำงานทันที บรรทัดที่ 30 สร้างปุ่มชื่อ `B2` มีข้อความว่า "Delete" เมื่อผู้ใช้คลิกปุ่มดังกล่าวโปรแกรมจะเรียกใช้ฟังชัน `deleteMember()` เข้ามาทำงาน ผลการทำงานแสดงดังรูปข้างบน

8. Menubutton

Menubutton คือ Widget ที่มีลักษณะเป็นเมนูแบบเลื่อนลงเมื่อผู้ใช้คลิกเลือก เมนูดังกล่าวจะคงอยู่ตลอดไปจนกว่าจะปิดโปรแกรม ผู้ใช้สามารถเลือกรายการได้รายการหนึ่งใน Menubutton โดยการคลิกที่รายการที่ต้องการ แสดงในรูปที่ 19.20



รูปที่ 19.20 แสดงรูปแบบของ Menubutton

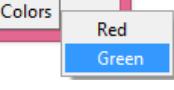
รูปแบบคำสั่งสำหรับการสร้าง Menubutton คือ

```
me = Menubutton( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Menubutton

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, activeforeground, anchor, bg, bitmap, bd, cursor, font, fg, height, width, image, justify, padx, pady, relief, state, text, textvariable, underline และ wraplength สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
direction	 กำหนดตำแหน่งของเมนูที่ต้องการแสดงผล ถ้ากำหนดเป็น LEFT เมนูจะปรากฏทางด้านซ้ายของปุ่มที่กำลังแสดงผล ถ้ากำหนดเป็น RIGHT เมนูจะปรากฏทางด้านขวาของปุ่ม ถ้ากำหนดเป็น "above" เมนูจะแสดงด้านบนของปุ่ม สำหรับค่าดีฟอลต์คือ "below" เช่น Menubutton(root, text="Colors", relief=RAISED, direction=RIGHT)
disabledforeground	สีของตัวอักษรใน Menubutton จะไม่ถูกใช้งาน เช่น Menubutton(root, text="Colors", disabledforeground="black")
menu	 กำหนดเมนูย่อยที่สัมพันธ์กับ Menubutton เช่น mb = Menubutton(root, text="Colors", relief=RAISED) mb.menu.add_checkbutton(label="Red") mb.menu.add_checkbutton(label="Green")

สำหรับตัวอย่างการสร้างและใช้งาน Menubutton แสดงในโปรแกรมตัวอย่างที่ 19.19 ดังนี้

Program Example 19.19: Menubutton

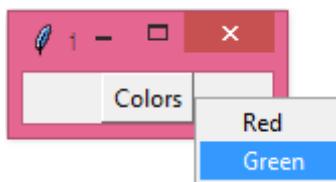
```
1 from tkinter import *
2 import tkinter
3
```

```

4   root = Tk()
5
⇒6 mb = Menubutton(root, text="Colors", relief=RAISED,
direction=RIGHT)
6   mb.grid()
⇒7 mb.menu = Menu(mb, tearoff=0)
⇒8 mb["menu"] = mb.menu
9
10  mb.menu.add_checkbutton(label="Red")
⇒11 mb.menu.add_checkbutton(label="Green")
11
12  mb.pack()
13
14  root.mainloop()
15

```

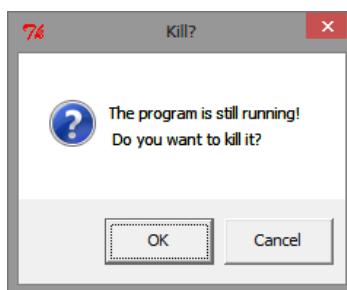
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.19 แสดงการสร้างและใช้งาน Menubutton บรรทัดที่ 6 สร้าง Menubutton ชื่อ mb โดยเมนูดังกล่าวมีข้อความคือ "Colors" และเมนูย่อยจะปรากฏทางด้านขวาของเมนู "Colors" บรรทัดที่ 8 สร้างเมนูย่อยภายใน Menubutton บรรทัดที่ 11 และ 12 เพิ่มรายการในเมนูย่อยชื่อ "Red" และ "Green" ตามลำดับ

9. Message (ข้อความ)

Message คือ Widget ที่มีลักษณะเป็นข้อความเพื่ออธิบายบางสิ่งบางอย่างในโปรแกรม (โดยแก้ไขไม่ได้) คล้ายกับ Label แต่แตกต่างกันคือ Message จะถูกปรับขนาดการแสดงผลให้เหมาะสมโดยอัตโนมัติ แสดงในรูปที่ 19.21



รูปที่ 19.21 แสดงรูปแบบของ Message

รูปแบบคำสั่งสำหรับการสร้าง Message คือ

```
me = Message( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Message

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ anchor, bg, bitmap, bd, cursor, font, fg, heigh, image, justify, padx, pady, relief, text, textvariable, underline, width และ wraplength

สำหรับตัวอย่างการสร้างและใช้งาน Message แสดงในโปรแกรมตัวอย่างที่ 19.20 ดังนี้

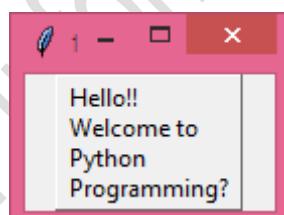
Program Example 19.20: Message

```

1  from tkinter import *
2
3  root = Tk()
4
5  var = StringVar()
6  me = Message(root, textvariable=var, relief=RAISED )
7
8  var.set("Hello!! Welcome to Python Programming?")
9  me.pack()
10 root.mainloop()

```

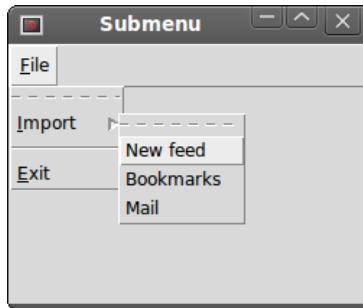
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.20 แสดงการสร้างและใช้งาน Message บรรทัดที่ 5 สร้างตัวแปรชื่อ var เป็นชนิดสตริง ใช้สำหรับเก็บข้อมูลของ Message บรรทัดที่ 6 สร้าง Message ชื่อ me โดยไม่มีข้อความใดๆ แสดงออกจนกว่าพิมพ์คุณการแสดงผลจะขึ้นอยู่กับตัวแปรที่ถูกกำหนดใน textvariable นั้นคือ ตัวแปรชื่อ var นั่นเอง บรรทัดที่ 8 กำหนดข้อความใหม่ว่า "Hello!! Welcome to Python Programming?" โดยใช้เมธอด set() ให้กับตัวแปร var ส่งผลให้ Message เปลี่ยนเป็นข้อความใหม่ที่กำหนดขึ้นทันที

10. Menu

Menu คือ Widget ที่มีลักษณะเป็นเมนูย่อย แบ่งออกเป็น 3 ประเภทคือ เมนูแบบ pop-up, toplevel และ pull-down ตัวอย่างเช่น เมนู File, Edit, Option, Windows และ Help เป็นต้น ดังแสดงในรูปที่ 19.22



รูปที่ 19.22 แสดงรูปแบบของ Menu

รูปแบบคำสั่งสำหรับการสร้าง Menu คือ

```
me = Menu( root, option=value, ... )
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Menu

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, activeforeground, bg, bd, cursor, disabledforeground, font, fg, relief, image สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
activeborderwidth	กำหนดขนาดความกว้างของกรอบเมนู เมื่อผู้ใช้คลิกเลือกเมนู (ดีฟอลต์เท่ากับ 1 พิกเซล) เช่น Menu(root, activeborderwidth=5)
postcommand	กำหนดให้เมนูเรียกใช้เมธอดหรือฟังชัน เมื่อมีผู้ใช้คลิกเลือกเมนูดังกล่าว เช่น Menu(root, postcommand=donothing)
selectcolor	กำหนดสีของปุ่ม checkbox หรือ radiobutton เมื่อปุ่มเหล่านี้ถูกเลือก เช่น Menu(root, selectcolor="red")
tearoff	โดยปกติ เมื่อเพิ่มรายการของเมนูย่อยเข้าไปในเมนูหลักจะเพิ่มในตำแหน่งที่ 1 แต่เมื่อกำหนดให้ tearoff=0 จะสามารถเพิ่มเมนูย่อยในตำแหน่งที่ 0 ได้ และเมนู

	ย่อไปนี้จะสามารถแสดงผลเป็นอิสระจากเมนูหลักได้ เช่น Menu(menuBar, tearoff=1)
title	กำหนดชื่อความ title ให้กับ Menu Widget

Widget ชนิด Menu มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด `add_command(options)` สร้างเมนูย่อยในเมนูหลัก เช่น


```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
```
- เมธอด `add_radiobutton(options)` สร้างเมนูชนิด checkbutton ในเมนูหลัก เช่น


```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_radiobutton(label="Exit", selectcolor="red")
```
- เมธอด `add_checkbutton(options)` สร้างเมนูชนิด checkbutton ในเมนูหลัก เช่น


```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_checkbutton(label="Exit", selectcolor="red")
```
- เมธอด `add_cascade(options)` เพิ่มชุดของเมนูย่อยที่เรียงต่อเนื่องกันในเมนูหลัก เช่น


```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=donothing)
menubar.add_cascade(label="File", menu=filemenu)
```
- เมธอด `add_separator()` สร้างเส้นเพื่อแยกเมนูย่อยออกจากกัน เช่น


```
filemenu.add_separator()
```
- เมธอด `add(type, options)` เพิ่มเมนูย่อยเข้าไปยังเมนูหลักแบบต่อท้าย (append) โดย type คือ ชนิดของเมนูต่างๆ เช่น cascade (submenu), checkbutton, radiobutton, หรือ separator และ options เช่น font, foreground หรือ image เป็นต้น
- เมธอด `delete(startindex [,endindex])` ลบช่วงรายการในเมนูย่อย โดยระบุรายการแรกใน startindex และรายการตัวสุดท้ายใน endindex เช่น


```
editmenu.delete(2, 4)
```
- เมธอด `entryconfig(index, options)` เมื่อเมนูย่อยถูกสร้างขึ้นแล้ว สามารถแก้ไขคุณสมบัติของ เมนูย่อยเหล่านั้น ผ่านเมธอด entryconfig() โดยอ้างด้วย index เช่น


```
filemenu.entryconfig(1, label="Test")
```
- เมธอด `index(item)` ส่งค่ากลับเป็นตำแหน่งของเมนูย่อยที่เลือก เช่น

- ```
filemenu.index(2)
```
- เมธอด `insert_separator(index)` เพิ่มเส้นสำหรับแบ่งเมนูย่อย โดยการระบุตำแหน่งที่ต้องการแทรกเส้นดังกล่าว เช่น  
`filemenu.insert_separator(2)`
  - เมธอด `invoke(index)` เรียกใช้คำสั่งที่มีหน้าที่เกี่ยวข้องกับเมนูที่เรียกใช้งาน ถ้าเป็นเมนูแบบ `checkbutton` เมธอดดังกล่าวจะทำหน้าที่สลับระหว่างปุ่มถูกเลือก (`set`) หรือถูกยกเลิก (`cleared`) ถ้าเมนูเป็นแบบ `radiobutton` จะถูกเช็คหรือยกเลิก เช่น  
`filemenu.invoke(2)`
  - เมธอด `type(index)` ส่งค่ากลับเป็นชนิดของเมนูย่อย เช่น `cascade`, `"checkbutton"`, `"command"`, `"radiobutton"`, `"separator"`, or `"tearoff"` เป็นต้น  
`filemenu.type(2)`

สำหรับตัวอย่างการสร้างและใช้งาน Menu แสดงในโปรแกรมตัวอย่างที่ 19.21 ดังนี้

#### Program Example 19.21: Menu

```

1 from tkinter import *
2
3 def donothing():
4 filewin = Toplevel(root)
5 button = Button(filewin, text="Do nothing button")
6 button.pack()
7
8 root = Tk()
9 menubar = Menu(root)
10 #To create File menu
11 filemenu = Menu(menubar, tearoff=0)
12 filemenu.add_command(label="New", command=donothing)
13 filemenu.add_command(label="Open", command=donothing)
14 filemenu.add_command(label="Save", command=donothing)
15 filemenu.add_command(label="Save as...", command=donothing)
16 filemenu.add_command(label="Close", command=donothing)
17 filemenu.add_separator()
18 filemenu.add_command(label="Exit", command=root.destroy)
19
20 menubar.add_cascade(label="File", menu=filemenu)
21
22 #To create Edit menu
23 editmenu = Menu(menubar, tearoff=0)
24 editmenu.add_command(label="Undo", command=donothing)
25 editmenu.add_separator()
26 editmenu.add_command(label="Cut", command=donothing)
27 editmenu.add_command(label="Copy", command=donothing)
28 editmenu.add_command(label="Paste", command=donothing)
29 editmenu.add_command(label="Delete", command=donothing)
30 editmenu.add_command(label="Select All", command=donothing)
31
32 menubar.add_cascade(label="Edit", menu=editmenu)
33

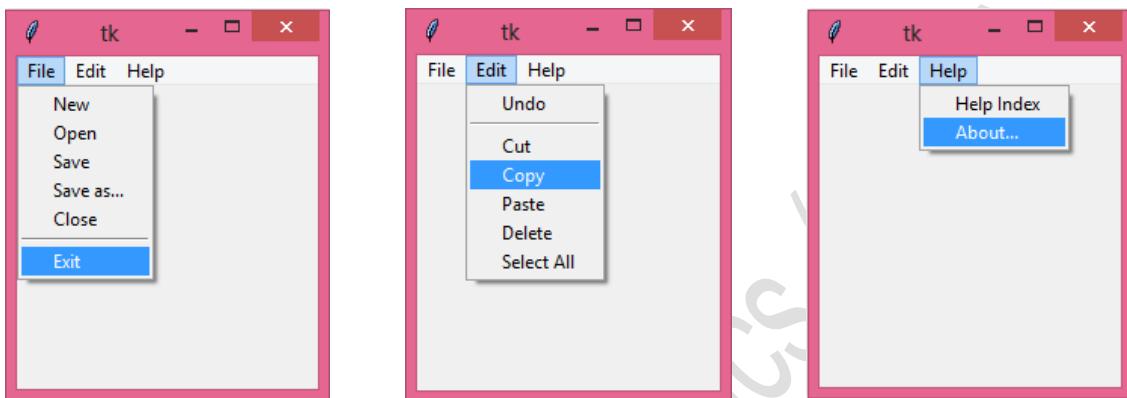
```

```

34 #To create Help menu
35 helpmenu = Menu(menubar, tearoff=0)
36 helpmenu.add_command(label="Help Index", command=donothing)
37 helpmenu.add_command(label="About...", command=donothing)
38
39 menubar.add_cascade(label="Help", menu=helpmenu)
40
41 root.config(menu=menubar)
42 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.21 แสดงการสร้างและใช้งานเมนู บรรทัดที่ 5 สร้างฟังชันชื่อว่า `donothing()` ทำหน้าที่สร้างหน้าต่างวินโดว์สีใหม่ที่เป็นอิสระจากวินโดว์หลัก โดยฟังชันดังกล่าวสร้างปุ่มและพิมพ์ข้อความบนปุ่มว่า "Do nothing button" บรรทัดที่ 9 สร้างวินโดว์หลักพร้อมกับเมนูหลักชื่อว่า `menubar` เพื่อใช้สำหรับรองรับเมนูย่อยที่จะสร้างขึ้นในคำสั่งลำดับถัดไป

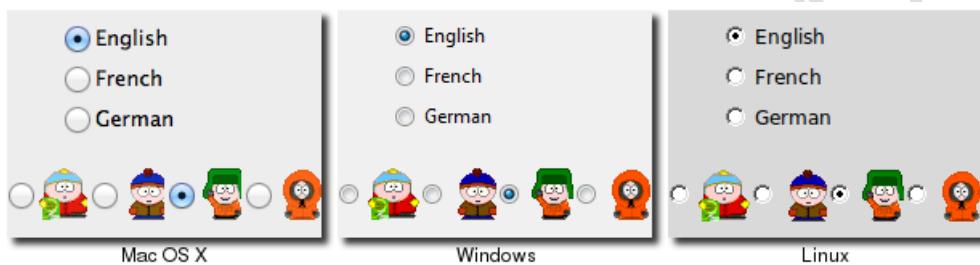
บรรทัดที่ 11 สร้างเมนูย่อยชุดแรกมีชื่อว่า `filemenu` บนเมนูหลัก (Menubar) โดยเมนูย่อยดังกล่าวจะถูกเพิ่มในตำแหน่งแรกของเมนูหลักได้ (`tearoff=0`) บรรทัดที่ 12 – 16 สร้างรายการของเมนูย่อยโดยเริ่มจาก "New", "Open", "New", "Save", ..., "Close" ตามลำดับ บรรทัดที่ 17 สร้างเส้นสำหรับแบ่งหมวดหมู่ของเมนูย่อยออกจากกัน บรรทัดที่ 18 สร้างเมนูย่อยชื่อว่า "Exit" เมื่อผู้ใช้งานกดปุ่มดังกล่าว โปรแกรมจะยุติการทำงานทันที (`command=root.destroy`) บรรทัดที่ 20 เพิ่มเมนูย่อยที่สร้างขึ้นชื่อว่า `filemenu` เข้าไปยังเมนูหลัก ในลักษณะแบบลำดับชั้นคล้ายน้ำตก (Cascade) เมนูหลักดังกล่าวมีชื่อว่า "File"

บรรทัดที่ 23 สร้างเมนูหลักชื่อว่า "Edit" เพื่อรองรับเมนูย่อยๆ ที่ทำหน้าที่เกี่ยวกับการแก้ไข แฟ้ม เช่น "Cut", "Copy", "Paste" และเส้นแยก (separator) ตามลำดับ (บรรทัดที่ 24 - 30) ในบรรทัดที่ 32 เพิ่มรายการเมนูย่อยที่สร้างขึ้นในเมนูหลักชื่อ "Edit" ด้วยเมธอด `add_cascade(label="Edit", menu=editmenu)` บรรทัดที่ 35 – 39 สร้างเมนูย่อยที่ทำหน้าที่เกี่ยวกับการช่วยเหลือ (Help) คำสั่งที่ใช้งานจะคล้ายกับการสร้างเมนู File และ Edit บรรทัดที่ 41 เป็นการกระตุ้นให้วินโดว์หลักอัพเดตเมนู

หลัก เพื่อให้ทุกอย่างทำงานได้อย่างถูกต้องด้วยเมธอด `root.config(menu=menubar)` ซึ่งจะต้องกระทำทุกครั้งเมื่อสร้างเมนูต่างๆ เสร็จเรียบร้อยแล้ว

## 11. Radiobutton

Radiobutton คือ Widget ที่มีลักษณะเป็นปุ่มกลมมีช่องว่างอยู่ภายใน เมื่อถูกเลือกจะเปลี่ยนสถานะเป็นสีที่เทียบขึ้น Radiobutton ส่วนใหญ่จะถูกสร้างเป็นกลุ่มของตัวเลือก เพื่อให้ผู้ใช้งานสามารถเลือกรายการได้รายการหนึ่งเพียงรายการเดียวเท่านั้น เช่น เลือกคำนำหน้าชื่อชาย, นางสาว, นาง เป็นต้น ดังแสดงในรูปที่ 19.23



รูปที่ 19.23 แสดงรูปแบบของ Radiobutton บนระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Radiobutton คือ

```
r = Radiobutton(root, option=value, ...)
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Radiobutton

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, activeforeground, anchor, bg, bitmap, borderwidth, command, cursor, font, fg, height, highlightbackground, highlightcolor, image, justify, padx, pady, relief, selectcolor, selectimage, state, text, textvariable, underline, width, wraplength สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

| Option | คำอธิบาย                                                                                                                                                                                                                               |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| value  | กำหนดค่าให้กับ Radiobutton และจะสัมพันธ์กับ variable เช่น ถ้ากำหนดค่าใน value เท่ากับจำนวนเต็ม (int) ตัวแปร variable จะต้องเป็นจำนวนเต็มด้วย ในกรณีของสตริงมีลักษณะเช่นเดียวกัน เช่น<br><code>var = IntVar(), str = StringVar()</code> |

|          |                                                                                                                                       |
|----------|---------------------------------------------------------------------------------------------------------------------------------------|
|          | <pre>Radiobutton(root, text="Mr.", variable=var, value=1) Radiobutton(root, text="Mr.", variable=str, value="Mr")</pre>               |
| variable | กำหนดตัวแปรที่ใช้สำหรับเก็บข้อมูลที่เกิดขึ้นจากการทำงานของ Radiobutton (ใช้ได้ทั้ง стрิงและจำนวนเต็ม) สำหรับตัวอย่างเมื่อมี昂กับ value |

Widget ชนิด Radiobutton มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด deselect() เคลียร์ค่ารายการที่เลือกไว้ใน Radiobutton
- เมธอด flash() วัด Radiobutton ใหม่
- เมธอด select() กำหนดค่าให้กับ Radiobutton

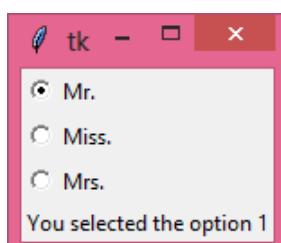
สำหรับตัวอย่างการสร้างและใช้งาน Radiobutton แสดงในโปรแกรมตัวอย่างที่ 19.22 ดังนี้

#### Program Example 19.22: Radiobutton

```

1 from tkinter import *
2
3 def sel():
4 selection = "You selected the option " + str(var.get())
5 label.config(text = selection)
6
7 root = Tk()
8 var = IntVar()
9 R1 = Radiobutton(root, text="Mr.", variable=var, value=1,
10 command=sel)
11 R1.pack(anchor = W)
12
13 R2 = Radiobutton(root, text="Miss.", variable=var, value=2,
14 command=sel)
15 R2.pack(anchor = W)
16
17 R3 = Radiobutton(root, text="Mrs.", variable=var, value=3,
18 command=sel)
19 R3.pack(anchor = W)
20
21 label = Label(root)
22 label.pack()
23 root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง

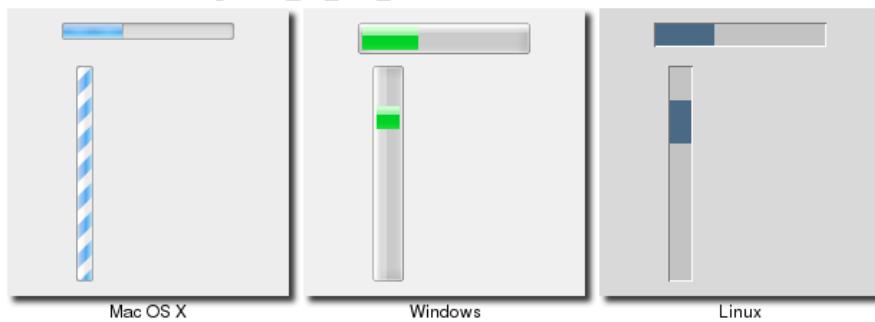


จากตัวอย่างโปรแกรมที่ 19.22 แสดงการสร้างและใช้งาน Radiobutton บรรทัดที่ 3 สร้างฟังชันชื่อว่า sel() ทำหน้าที่กำหนดข้อความใหม่ให้กับ Label เป็น "You selected the option" ตามด้วยค่าที่เก็บอยู่ในตัวแปร var (var.get()) โดยใช้เมธอด Label.config() บรรทัดที่ 8 สร้างตัวแปรชื่อ var เป็นชนิดจำนวนเต็ม สำหรับเก็บข้อมูลที่เกิดขึ้นจากการดำเนินการได้ บน Radiobutton บรรทัดที่ 9 สร้าง Radiobutton ชื่อ R1 มีข้อความว่า "Mr." มีค่าเท่ากับ 1 (value=1) เมื่อมีการคลิกเลือกปุ่ม Radiobutton ดังกล่าว ผลลัพธ์จาก value จะถูกนำมาเก็บไว้ในตัวแปรชื่อ var (variable=var) เมื่อปุ่มดังกล่าวถูกคลิกเลือก โปรแกรมจะเรียกฟังชัน ชื่อว่า sel() มาทำงานแทนที่ (command=sel)

บรรทัดที่ 12 และ 15 สร้าง Radiobutton ชื่อ R1 และ R2 มีข้อความว่า "Miss." และ "Mrs." โดยมีค่าเท่ากับ 2 และ 3 ตามลำดับ เมื่อคลิกเลือกปุ่มทั้งสอง โปรแกรมจะเรียกใช้งานฟังชัน sel() เช่นเดียวกับเมธุ R1 บรรทัดที่ 18 สร้าง Label ชื่อ label เพื่อแสดงผลลัพธ์จากการคลิกเลือกปุ่มใน Radiobutton ออกจอภาพ ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

## 12. Scale

Scale คือ Widget ที่มีลักษณะเลื่อนสไลด์ขึ้นลงหรือซ้ายขวาได้ เพื่อทำหน้าที่แสดงข้อมูลของข้อมูลที่ผู้ใช้ต้องการ เช่น ปรับขนาดความเข้มของสี ความสว่าง ความคมชัด เป็นต้น ดังแสดงในรูปที่ 19.24



รูปที่ 19.24 แสดงรูปแบบของ Scale บนระบบปฏิบัติการต่างๆ

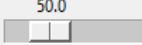
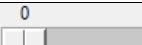
รูปแบบคำสั่งสำหรับการสร้าง Scale คือ

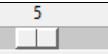
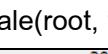
```
s = Scale(root, option=value, ...)
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Scale

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, bg, bd, command, cursor, font, fg, highlightbackground, highlightcolor, length, relief, state, variable, width สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

| Option       | คำอธิบาย                                                                                                                                                                                                                                                                                                                                                     |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| digits       |  เป็น Option ที่ใช้สำหรับแสดงรูปแบบของสเกล (scale) มี 3 รูปแบบ คือ จำนวนเต็ม (IntVar), จำนวนจริง (DoubleVar(float)), และสตริง (StringVar) เช่น Scale(root, <b>digits=4</b> , orient=HORIZONTAL)                                                                             |
| from_        |  เป็นเลขจำนวนเต็ม หรือจำนวนจริงที่ใช้กำหนดขอบเขตเริ่มต้นของ สเกล เช่น Scale(root, <b>from_=0</b> , to=200, orient=HORIZONTAL)                                                                                                                                               |
| label        |  แสดงข้อความกำกับสเกล ข้อความจะปรากฏที่มุมด้านซ้ายบนเมื่อ สเกลเป็นชนิดแนวโน้ม, ข้อความจะปรากฏมุมด้านขวาบนเมื่อสเกล เป็นชนิดแนวตั้ง ค่าดีฟอลต์จะไม่แสดงข้อความ เช่น Scale(root, <b>label="Scale"</b> , orient=HORIZONTAL)                                                    |
| orient       |  กำหนด orient=HORIZONTAL เมื่อต้องการให้สเกลวางอยู่ใน แนวอน (แนวแกน x) และ orient=VERTICAL เมื่อต้องการสร้างสเกลในแนวตั้ง (แกน y) ค่าดีฟอลต์เป็นสเกลในแนวอน (HORIZONTAL) เช่น Scale(root, <b>orient=HORIZONTAL</b> )                                                       |
| repeatdelay  |  ใช้กำหนดเวลาเพื่อหน่วงการเคลื่อนที่ของปุ่มในสเกล (เคลื่อนที่ช้า-ลง) ในกรณีที่ผู้ใช้คลิกในช่องของสเกลค้างไว้ (ดีฟอลต์=300) เช่น Scale(root, <b>repeatdelay=5</b> , orient=HORIZONTAL)                                                                                     |
| resolution   |  กำหนดช่วงของสเกลเมื่อเพิ่มขึ้นหรือลดลง เช่น เมื่อกำหนดช่วงของ สเกลเท่ากับ from_=-1.0 ถึง to=1.0 และกำหนด resolution=0.5 สเกลจะเพิ่มขึ้นและ ลดลงดังนี้คือ -1.0, -0.5, 0.0, +0.5, และ +1.0 เช่น Scale(root, from_=-1.0, to=1.0, <b>resolution=0.5</b> , orient=HORIZONTAL) |
| showvalue    |  โดยปกติค่าของสเกลจะแสดงผลร่วมกับแท็บสเกลเสมอ เมื่อไม่ ต้องการแสดงค่าของสเกลให้กำหนด showvalue=0 เช่น Scale(root, from_=-1.0, to=1.0, resolution=0.5, <b>showvalue=0</b> , orient=HORIZONTAL)                                                                             |
| sliderlength |  กำหนดขนาดของแท็บสไลด์ของสเกล (โดยปกติแท็บจะมีขนาดเท่ากับ 30 พิกเซล) เช่น Scale(root, <b>sliderlength=10</b> , orient=HORIZONTAL)                                                                                                                                         |
| takefocus    | โดยปกติ สเกลจะโฟกัสเป็นแบบวงรอบ เมื่อไม่ต้องการพุติกรรมดังกล่าวให้กำหนด takefocus=0 เช่น Scale(root, from_=1, to=10, resolution=1, <b>takefocus=0</b> )                                                                                                                                                                                                      |

|              |                                                                                                           |                                                                                                                                                                                          |
|--------------|-----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tickinterval | <br>1 2 3 4 5 6 7 8 9 10 | กำหนดการแสดงตัวเลขช่วงของสเกล เมื่อเป็นสเกลแนวอนจะแสดงช่วงสเกลด้านล่าง แต่ถ้าเป็นสเกลแนวตั้งจะแสดงด้านซ้าย เช่น<br><b>Scale(root, from_=1, to=10, tickinterval=1, orient=HORIZONTAL)</b> |
| to           | <br>200                  | เป็นเลขจำนวนเต็ม หรือจำนวนจริงที่ใช้กำหนดขอบเขตสิ้นสุดของ สเกล เช่น <b>Scale(root, from_=0, to=200, orient=HORIZONTAL)</b>                                                               |
| troughcolor  | <br>6                    | กำหนดสีของร่องหรือรากของสเกล เช่น <b>Scale(root, from_=1, to=10, troughcolor="red", orient=HORIZONTAL)</b>                                                                               |

Widget ชนิด Scale มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

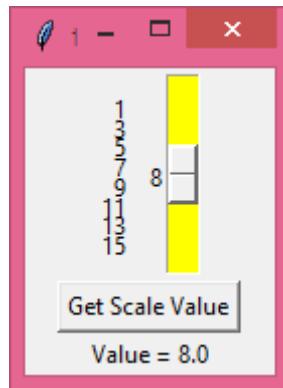
- เมื่อ调用 get() คืนค่าของสเกลปัจจุบันที่กำลังทำงานอยู่
  - เมื่อ调用 set(value) กำหนดค่าสเกลใหม่

สำหรับตัวอย่างการสร้างและใช้งาน Scale แสดงในโปรแกรมตัวอย่างที่ 19.23 ดังนี้

### Program Example 19.23: Scale

```
1 from tkinter import *
2
3 def sel():
4 selection = "Value = " + str(var.get())
5 label.config(text = selection)
6
7 root = Tk()
8 var = DoubleVar()
9 scale = Scale(root, from_=1, to=15, resolution=1,
10 tickinterval=1, troughcolor="yellow", variable=var,
11 orient=VERTICAL)
12 scale.pack(anchor=CENTER)
13
14 button = Button(root, text="Get Scale Value", command=sel)
15 button.pack(anchor=CENTER)
16
17 label = Label(root)
18 label.pack()
19
20 root.mainloop()
```

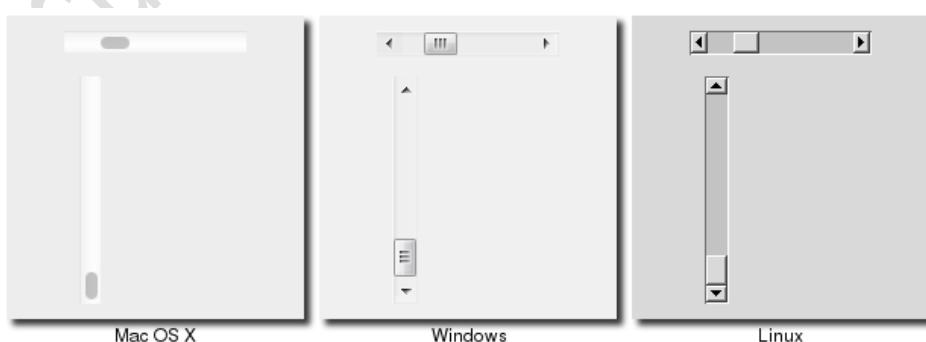
ผลลัพธ์ที่ได้เมื่อส่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.23 แสดงการสร้างและใช้งาน Scale บรรทัดที่ 3 สร้างฟังชันชื่อว่า sel() ทำหน้าที่แสดงข้อมูลให้กับ Label มีค่าเท่ากับ "Value =" ตามด้วยค่าที่เก็บอยู่ในตัวแปร var (var.get()) โดยใช้เมธอด Label.config() บรรทัดที่ 8 สร้างตัวแปรชื่อ var เป็นชนิดจำนวนจริงขนาดใหญ่ (Double) สำหรับเก็บข้อมูลที่เกิดขึ้นจากการเลื่อนสเกล บรรทัดที่ 9 สร้าง Scale ชื่อ scale มีขอบเขตของสเกล ตั้งแต่ 1 ถึง 15 (from\_=1, to=15), ช่วงของสเกลเท่ากับ 1 (resolution=1), แสดงค่าของสเกลอยู่ด้านซ้ายตั้งแต่ 1 ถึง 15 (tickinterval=1), ร่องของสเกลเป็นสีเหลือง (troughcolor="yellow"), เป็นสเกลในแนวตั้ง (orient=VERTICAL), เมื่อผู้ใช้เลื่อนแท็บของสเกล ผลลัพธ์จะเก็บไว้ในตัวแปรชื่อ var (variable=var) บรรทัดที่ 12 สร้างปุ่มชื่อ button มีข้อความว่า "Get Scale Value" เมื่อคลิกปุ่มดังกล่าว โปรแกรมจะเรียกฟังชัน sel() มาทำงาน ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

### 13. Scrollbar

Scrollbar คือ Widget ที่มีลักษณะเป็นแท็บสไลด์เลื่อนขึ้น-ลง หรือซ้าย-ขวาได้ เพื่อเพิ่มขนาดพื้นที่สำหรับแสดงผลหรือให้ผู้ใช้ป้อนข้อมูลเพิ่มขึ้น นิยมใช้งานร่วมกับ Listbox, Text, Canvas และ Entry เป็นต้น ดังแสดงในรูปที่ 19.25



รูปที่ 19.25 แสดงรูปแบบของ Scrollbar บนระบบปฏิบัติการต่างๆ

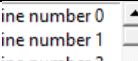
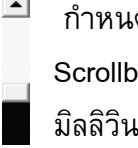
รูปแบบคำสั่งสำหรับการสร้าง Scrollbar คือ

```
s = Scrollbar(root, option=value, ...)
```

พารามิเตอร์คือ

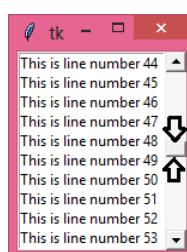
- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Scrollbar

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets (Scale) ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, bg, bd, command, cursor, orient, repeatdelay, takefocus, troughcolor width สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

| Option         | คำอธิบาย                                                                                                                                                                                                                                                                    |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| borderwidth    |  กำหนดขนาดของหัวลูกศรและแท็บสไลด์ของ Scrollbar เช่น Scrollbar(root, borderwidth=5)                                                                                                         |
| jump           | กำหนดพฤติกรรมเมื่อกดการเคลื่อนที่ของ Scrollbar (แท็บ) เมื่อกำหนด jump=0 จะทำให้สามารถเรียกใช้ command เพื่อเรียกใช้ฟังชันที่กำหนดไว้เข้ามารаботา, เมื่อกำหนด jump=1 จะปิดการใช้งาน command เช่น Scrollbar(root, jump=0, command=jumpCall)                                   |
| repeatinterval |  กำหนดระยะเวลาเมื่อผู้ใช้กดค้างที่รางของ Scrollbar ก่อนที่แท็บของ Scrollbar จะเคลื่อนที่ไปยังทิศทางที่ผู้ใช้ต้องการ ค่าเดิมอยู่ที่ 300 มิลลิวินาที เช่น Scrollbar(root, repeatdelay=100) |

Widget ชนิด Scale มีメธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด get() คืนค่าตำแหน่งปั๊จจุบันของแท็บที่กำลังทำงานอยู่ซึ่งมี 2 ค่าคือ (a, b) โดย a คือ ตำแหน่งที่อยู่ด้านซ้าย (Scrollbar เป็นชนิดแนวนอน) และด้านบน (Scrollbar เป็นชนิดแนวตั้ง) ของแท็บ และ b คือ ค่าตำแหน่งที่อยู่ด้านขวาและล่างของแท็บ เช่น print(scrollbar.get()) ผลลัพธ์คือ (0.41, 0.51) ⇒ เมื่อ Scrollbar เป็นแนวตั้ง ค่า 0.41 คือตำแหน่งของแท็บด้านบน และ 0.51 คือตำแหน่งของแท็บด้านล่าง ดังรูป



- เมธอด set(first, last) ใช้สำหรับกรณีที่นำ Scrollbar ไปใช้กับ Widget ชนิดอื่นๆ โดย Widgets ที่ต้องการเรียกใช้เมธอดดังกล่าวจะเรียกผ่าน xscrollcommand หรือ yscrollcommand แทน

ผลลัพธ์ที่ได้เมื่ออนกับการเรียกเมธอด get() นั้นเอง (สามารถอ่านการใช้งาน xscrollcommand หรือ yscrollcommand จากหัวข้อ Canvas, Entry หรือ Listbox)

สำหรับตัวอย่างการสร้างและใช้งาน Scrollbar แสดงในโปรแกรมตัวอย่างที่ 19.24 ดังนี้

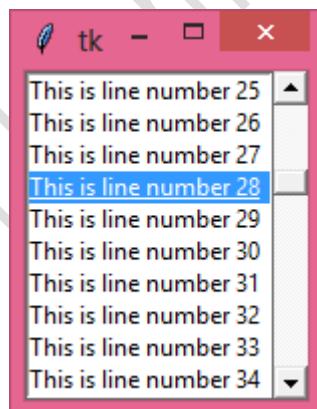
**Program Example 19.24: Scrollbar**

```

1 from tkinter import *
2
3 root = Tk()
4
5 scrollbar = Scrollbar(root)
6 scrollbar.pack(side = RIGHT, fill=Y)
7
8 mylist = Listbox(root, yscrollcommand = scrollbar.set)
9
10 for line in range(100):
11 mylist.insert(END, "This is line number " + str(line))
12
13 mylist.pack(side = LEFT, fill = BOTH)
14 scrollbar.config(command = mylist.yview)
15
16 mainloop()

```

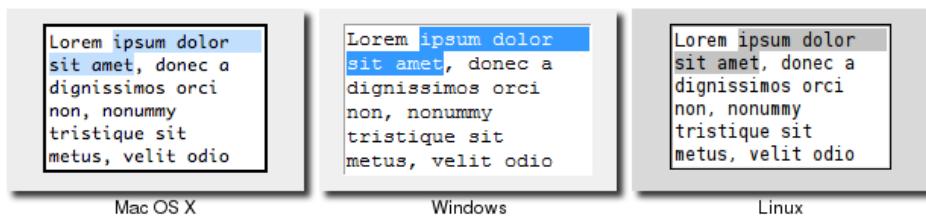
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.24 แสดงการสร้างและใช้งาน Scrollbar บรรทัดที่ 5 สร้าง Scrollbar ชื่อ scrollbar บรรทัดที่ 8 สร้าง Listbox ชื่อว่า mylist จากนั้นทำการเพิ่ม Scrollbar เข้าไปใน Listbox ในแนวตั้งหรือแนวแกน y ด้วยคำสั่ง yscrollcommand (yscrollcommand = scrollbar.set) บรรทัดที่ 10 โปรแกรมใช้ลูป for สั่งพิมพ์ "This is line number" และตามด้วยตัวเลขที่เริ่มตั้งแต่ 0 – 99 ลงใน Listbox โดยใช้เมธอด insert() ส่งผลให้ Scrollbar สร้างแท็บสไลด์มีขนาดที่ครอบคลุมรายการทั้งหมด (ถ้าสั่งพิมพ์รายการน้อยๆ เช่น 5 รายการ Scrollbar จะไม่สร้างแท็บสำหรับเลื่อนสไลด์ให้) บรรทัดที่ 14 โปรแกรมสั่งกระตุ้นให้ Scrollbar ทำงานด้วยเมธอด config() ผ่านอ้อมบล็อก command ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

## 14. Text

Text คือ Widget ที่อนุญาตให้ผู้เขียนโปรแกรมสามารถสร้างข้อความเพื่อขอรับข้อมูลสิ่งของอย่างในโปรแกรม โดย Text มีความสามารถหลายอย่าง เช่น เปลี่ยนสีพื้นข้อความ สีตัวอักษร รูปแบบฟอนต์ ขนาด และอื่นๆ ดังแสดงในรูปที่ 19.26



รูปที่ 19.26 แสดงรูปแบบของ Text บนระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Text คือ

```
t = Text(root, option=value, ...)
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Text

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, height, highlightbackground, highlightcolor, padx, pady, relief, state, width, xscrollcommand, yscrollcommand สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

| Option             | คำอธิบาย                                                                                                                                                                                                                                     |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| exportselection    | โดยปกติเมื่อผู้ใช้เลือกข้อความภายใน Text ข้อมูลที่ถูกเลือกจะถูกส่งไปเก็บไว้ในคลิปบอร์ด (Clipboard) โดยอัตโนมัติ เมื่อไม่ต้องการให้ข้อมูลดังกล่าวถูกส่งไปยังคลิปบอร์ด ให้กำหนด exportselection = 0 เช่น Text(root, <b>exportselection=0</b> ) |
| highlightthickness | กำหนดขนาดของ Highlight Focus ของ Text ค่าเดียวกันกับ 1 แต่ถ้าไม่ต้องการให้ออฟชันดังกล่าวทำงานให้กำหนดเป็น 0 เช่น Text(root, <b>highlightthickness=0</b> )                                                                                    |
| insertbackground   | หนดสีของของเครื่องเซอร์เซอร์ ณ ตำแหน่งปัจจุบัน เช่น Text(root, <b>insertbackground="red"</b> )                                                                                                                                               |

|                   |                                                                                                                                                                                                                                  |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| insertborderwidth | ขนาดของกรอบแบบ 3-D รอบๆ เคอร์เซอร์ ค่าดีฟอลต์คือ 0                                                                                                                                                                               |
| insertofftime     | Bye.....  กำหนดเวลาให้เคอร์เซอร์หยุดทำงาน มีหน่วยเป็นมิลลิวินาที ค่าดีฟอลต์เท่ากับ 300 มิลลิวินาที ถ้าไม่ต้องการให้เคอร์เซอร์กระพริบ กำหนดให้ insertofftime=0 เช่น Text(root, insertbackground="red", <b>insertofftime=100</b> ) |
| insertontime      | Bye.... กำหนดเวลาให้เคอร์เซอร์ทำงาน (กระพริบ) มีหน่วยเป็นมิลลิวินาที ค่าดีฟอลต์เท่ากับ 600 มิลลิวินาที ถ้าไม่ต้องการให้เคอร์เซอร์ปรากฏ กำหนดให้ insertontime=0 เช่น Text(root, insertbackground="red", <b>insertontime=0</b> )   |
| insertwidth       | Bye....  กำหนดขนาดของเคอร์เซอร์ ค่าดีฟอลต์เท่ากับ 2 พิกเซล เช่น Text(root, insertbackground="red", <b>insertwidth=10</b> )                                                                                                       |
| selectbackground  | Bye Bye... กำหนดสีพื้นหลังเมื่อเมาส์ไปเลือกข้อความ เช่น Text(root, insertbackground="red", <b>selectbackground="black"</b> )                                                                                                     |
| selectborderwidth | กำหนดความกว้างของกรอบรอบๆ Text                                                                                                                                                                                                   |
| spacing1          | Hello.. กำหนดขนาดความกว้างด้านบนของข้อความในแต่ละบรรทัด ค่าดีฟอลต์เท่ากับ 0 เช่น Text(root, <b>spacing1=10</b> )                                                                                                                 |
| spacing2          | กำหนดระยะห่างระหว่างบรรทัดของข้อความใน Text ค่าดีฟอลต์เท่ากับ 0 เช่น Text(root, <b>spacing2=10</b> )                                                                                                                             |
| spacing3          | Hello.. กำหนดขนาดความกว้างด้านล่างของข้อความในแต่ละบรรทัด ค่าดีฟอลต์เท่ากับ 0 เช่น Text(root, <b>spacing3=10</b> )                                                                                                               |
| tabs              | Hello  ... กำหนดขนาดของแท็บในข้อความ หน่วยเป็นพิกเซล เช่น Text(root, <b>tabs=100</b> ) (ให้ผู้ใช้คลิกที่ข้อความแล้วกดสوبกดที่ปุ่ม TAB)                                                                                           |
| wrap              | กำหนดขนาดของข้อความที่จะถูกครอบ (Focus) ถ้ากำหนด wrap=WORD โปรแกรมครอบทีละคำ (แยกด้วยข้อความว่าง) แต่ถ้ากำหนดเป็น wrap=CHAR จะเป็นการครอบทีละตัวอักษรแทน เช่น Text(root, <b>wrap=WORD</b> )                                      |
| xscrollcommand    | ed with Tkinter provided... กำหนด Scrollbar ให้กับ Text ในแนวนอน ใช้ในกรณี need to do is perform... ที่ข้อความมีขนาดความยาวมากๆ เช่น Text(root, <b>xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set</b> )            |
| yscrollcommand    | กำหนด Scrollbar ให้กับ Text ในแนวตั้ง เช่น Text(root, <b>xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set</b> )                                                                                                      |

Widget ชนิด Text มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด delete(startindex [,endindex]) ลบตัวอักษรหรือข้อความใน Text โดยระบุช่วงของข้อความ ถ้ากำหนดเฉพาะ startindex ข้อความจะถูกลบตั้งแต่ startindex ถึงตำแหน่งสิ้นสุด

ของข้อความ เช่น `text.delete(1.0)` โดย 1 หมายถึงบรรทัดที่ 1, 0 หมายถึงอักษรตัวที่ 0 (*line number.character number*)

- เมธอด `get(startindex [,endindex])` คืนค่าเป็นตัวอักษรหรือข้อความจาก Text โดยใช้ `startindex` กำหนดตำแหน่งเริ่มต้น และตำแหน่งสุดท้ายของข้อความด้วย `endindex` เช่น `print(text.get(1.0, END))`
- เมธอด `index(index)` คืนค่าสมบูรณ์ (absolute value) จาก Text โดยใช้ `index` กำหนดตำแหน่ง เช่น `print(text.index(4.0))`

Text ยังมีเมธอดที่ช่วยสำหรับการทำ Marks, Tabs และ Indexes ดังนี้

- เมธอด `index(mark)` คืนค่าแถวและคอลัมน์ที่ผู้ใช้ทำเครื่องหมายไว้ (Marks) เช่น `text.mark_set("Python", INSERT)`  
`print(text.index(INSERT))`  
ผลลัพธ์ที่ได้คือ 1.26 (แถวที่ 1, คอลัมน์ที่ 26)
- เมธอด `mark_gravity(mark [,gravity])` กำหนดลักษณะทิศทางของการ Marks ซึ่งเป็นไปได้ 2 ทิศทาง คือ ทิศทางที่เป็นจุดอ้างอิงทางด้านขวา (RIGHT) หรือด้านซ้าย (LEFT) เช่น ข้อความ "This is Python Programming" เมื่อทำการ Marks ไว้กับข้อความเป็นคำว่า "Python" และทำการกำหนดทิศทางเป็น LEFT แสดงรูปแบบคำสั่งดังนี้  
`text.insert(INSERT, "This is Python Programming")`  
`text.mark_set("Python", INSERT)`  
`text.mark_gravity("Python", LEFT)`
- เมธอด `mark_names()` คืนค่าข้อความทั้งหมดที่ Marks ไว้ เช่น `print(text.mark_names())`
- เมธอด `mark_set(mark, index)` กำหนดตำแหน่ง Marks ใหม่ เช่น  
`text.insert(INSERT, "This is Python Programming")`  
`text.mark_set("Python", CURRENT)`
- เมธอด `mark_unset(mark)` เคลียร์ค่าที่ Marks ไว้ เช่น `text.mark_unset(CURRENT)`

Text ยังมีเมธอดที่ช่วยสำหรับการทำงานเกี่ยวกับ Tag ดังนี้

- เมธอด `tag_add(tagname, startIndex[, endIndex] ...)` ทำหน้าที่กำหนดป้ายชื่อลงในข้อความใน Text โดย `tagname` คือชื่อของ tag, `startIndex` คือตำแหน่งเริ่มต้นที่ต้องการเพิ่ม tag, `endIndex` (option=ไม่ใส่ก็ได้) ตำแหน่งสุดท้ายที่ต้องการเพิ่ม tag เช่น  
`text.tag_add("here", "1.0", "1.4")`

จากคำสั่งด้านบนเป็นการกำหนด tag ชื่อ "here" โดยขนาดของ tag มีความยาวตั้งแต่ตัวอักษรที่ 0 (.0) ถึง 4 (.4) และอยู่ในบรรทัดที่ 1 (1.) ของข้อความใน Text

- เมธอด tag\_config ทำหน้าที่ปรับแต่งคุณสมบัติต่างๆ หลังจากที่สร้าง Text ขึ้นมาใช้งานแล้ว เช่น การกำหนดสีพื้นหลัง สีตัวอักษร ขัดเส้นได้ เป็นต้น เช่น
 

```
text.tag_config("here", background="yellow", foreground="blue")
```
- เมธอด tag\_delete(tagname) ทำหน้าที่ลบ tag จากที่เคยกำหนดไว้
- เมธอด tag\_remove(tagname [,startindex[,endindex]] ...) ทำหน้าที่ลบ tag ออกจากพื้นที่ที่เคยกำหนดไว้ แต่ไม่ลบ Tag ที่นิยามเอาไว้ในโปรแกรม

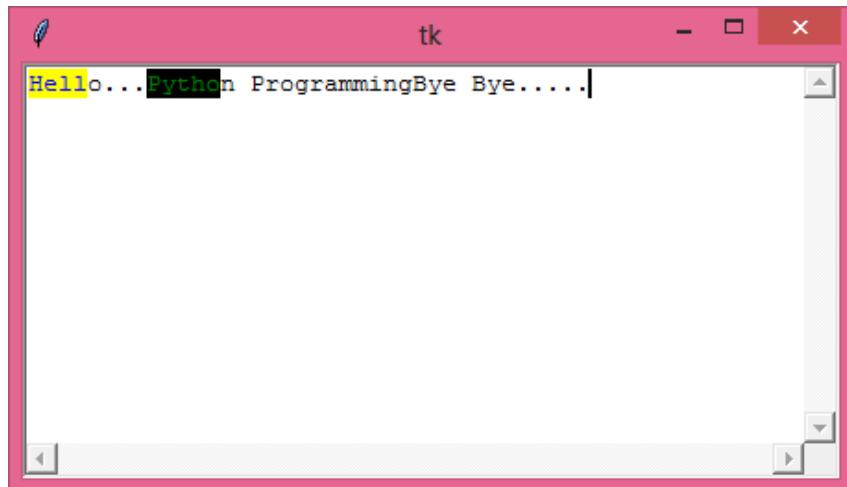
สำหรับตัวอย่างการสร้างและใช้งาน Text แสดงในโปรแกรมตัวอย่างที่ 19.25 ดังนี้

#### Program Example 19.25: Text

```

1 from tkinter import *
2
3 root = Tk()
4 frame = Frame(root, bd=2, relief=SUNKEN)
5
6 frame.grid_rowconfigure(0, weight=1)
7 frame.grid_columnconfigure(0, weight=1)
8
9 xscrollbar = Scrollbar(frame, orient=HORIZONTAL)
10 xscrollbar.grid(row=1, column=0, sticky=E+W)
11
12 yscrollbar = Scrollbar(frame)
13 yscrollbar.grid(row=0, column=1, sticky=N+S)
14
15 text = Text(frame, bd=0, xscrollcommand=xscrollbar.set,
16 yscrollcommand=yscrollbar.set)
17 text.grid(row=0, column=0, sticky=N+S+E+W)
18
19 xscrollbar.config(command=text.xview)
20 yscrollbar.config(command=text.yview)
21
22 text.insert(INSERT, "Hello...Python Programming")
23 text.insert(END, "Bye Bye.....")
24 text.tag_add("here", "1.0", "1.4")
25 text.tag_add("start", "1.8", "1.13")
26 text.tag_config("here", background="yellow",
27 foreground="blue")
28 text.tag_config("start", background="black",
29 foreground="green")
30
31 frame.pack()
```

ผลลัพธ์ที่ได้มี่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.25 แสดงการสร้างและใช้งาน Text บรรทัดที่ 4 สร้างเฟรมชื่อ frame มีความหนาของกรอบเท่ากับ 2 พิกเซล เป็นชนิดกรอบแบบร่องลึก บรรทัดที่ 6 และ 7 สร้างกริดบนเฟรมมีขนาดเท่ากับ 1 แถว 1 คอลัมน์ บรรทัดที่ 9 และ 10 สร้าง Scrollbar ในแนวนอนลงบนเฟรม วางในตำแหน่งแถวที่ 1 และคอลัมน์ที่ 0 ในกริด และวาง Scrollbar จากด้านทิศตะวันออกไปทิศตะวันตก (sticky=E+W) บรรทัดที่ 12 และ 13 สร้าง Scrollbar ในแนวตั้งลงบนเฟรม วางในตำแหน่งแถวที่ 0 และคอลัมน์ที่ 1 ในกริด และวาง Scrollbar จากด้านทิศเหนือไปทิศใต้ (sticky=N+S)

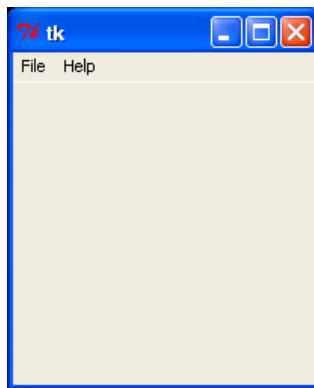
บรรทัดที่ 15 สร้าง Text ลงบนเฟรม โดยมี Scrollbar อ่ายด้านขวา (yscrollcommand) และด้านล่าง (xscrollcommand) ของ Text ในบรรทัดที่ 16 Text จะถูกวางลงในแถวที่ 0 คอลัมน์ 0 ของกริด ใน 4 ทิศทาง (row=0, column=0, sticky=N+S+E+W) บรรทัดที่ 18 และ 19 ออกคำสั่งให้ scrollbar แสดงผลใน Text

บรรทัดที่ 21 เพิ่มข้อความว่า "Hello...Python Programming" ลงใน Text โดยใช้ตำแหน่งที่ Marks ไว้คือ INSERT (โดยปกติ INSERT Marks จะอยู่ในตำแหน่งด้านท้ายของข้อความใน Text สามารถเปลี่ยนจุด Marks ของ INSERT ใหม่ได้โดยใช้เมธอด mark\_set() ที่อธิบายไว้ในหัวข้อที่ผ่านมา) บรรทัดที่ 22 เพิ่มข้อความว่า "Bye Bye....." ลงใน Text โดยใช้ตำแหน่งที่ Marks ไว้คือ END ซึ่งเป็นตำแหน่งสุดท้ายในข้อความ ผลลัพธ์จากการทำงานของบรรทัดที่ 21 และ 22 คือข้อความว่า "Hello...Python Programming Bye Bye....." บน Text

บรรทัดที่ 23 สร้าง tag ชื่อ "here" โดยครอบคอลัมน์ตัวอักษรตั้งแต่ตัวที่ 0 – 3 ของบรรทัดที่ 1 เอาไว้ ("here", "1.0", "1.4") คือข้อความว่า "Hell" บรรทัดที่ 24 สร้าง tag ชื่อ "start" โดยครอบคอลัมน์ตัวอักษรตั้งแต่ตัวที่ 8 – 13 เอาไว้ คือข้อความว่า "Phton" บรรทัดที่ 25 สั่งให้ระบายสี tag ชื่อ "here" โดยมีสีของพื้นหลังเป็นสีเหลือง และตัวอักษรเป็นสีน้ำเงิน บรรทัดที่ 26 สั่งให้ระบายสี tag ชื่อ "start" โดยมีสีของพื้นหลังเป็นสีดำ และตัวอักษรเป็นสีเขียวตามลำดับ ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

## 15. Toplevel

Toplevel เป็น Widget ที่อยู่บนสุดของวินโดว์ไม่จำเป็นต้องมีวินโดว์อื่นๆ อยู่ควบคุณหรืออยู่ภายใต้วินโดว์ใดๆ หรือพอดีๆ คือ Toplevel จะถูกดูแลจาก Window Manager โดยตรงนั่นเอง ดังแสดงในรูปที่ 19.27



รูปที่ 19.27 แสดงรูปแบบของ Toplevel

รูปแบบคำสั่งสำหรับการสร้าง Toplevel คือ

```
t = Toplevel(root, option=value, ...)
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Toplevel

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, height, relief, width เป็นต้น

Widget ชนิด Toplevel มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด **deiconify()** แสดงวินโดว์ทันที เมื่อเรียกใช้เมธอดดังกล่าว เช่น  
`top = Toplevel()`  
`top.deiconify()`
- เมธอด **frame()** คืนค่า system-specific window identifier เช่น  
`print(text.get(1.0, END))`  
 ผลลัพธ์คือ 0x2403c2
- เมธอด **group(window)** จัดกลุ่มของวินโดว์

- เมธอด iconify() แปลงวินโดวส์ไปเป็น icon
- เมธอด protocol(name, function) ลงทะเบียนฟังชันเพื่อทำหน้าที่เป็น callback ฟังชัน
- เมธอด state() คืนค่าสถานะปัจจุบันของวินโดวส์ ค่าที่เป็นไปได้คือ Normal, iconic, withdrawn และ icon
- เมธอด withdraw() ลบวินโดวส์ออกจากจอภาพ
- เมธอด maxsize(width, height) กำหนดขนาดวินโดวส์ที่ใหญ่ที่สุด
- เมธอด minsize(width, height) กำหนดขนาดวินโดวส์ที่เล็กที่สุด
- เมธอด title(string) กำหนด title ของวินโดวส์ที่สร้างขึ้น

สำหรับตัวอย่างการสร้างและใช้งาน Text แสดงในโปรแกรมตัวอย่างที่ 19.27 ดังนี้

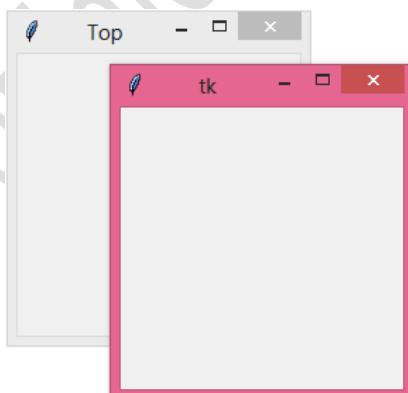
**Program Example 19.26: Toplevel**

```

1 from tkinter import *
2
3 ⇒3 root = Tk()
4 ⇒4 top = Toplevel()
5 ⇒5 top.title("Toplevel")
6 ⇒6 top.deiconify()
7
8 top.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.26 แสดงการสร้างและใช้งาน Toplevel บรรทัดที่ 3 สร้างวินโดวส์หลักหรือ root window ซึ่งว่า root บรรทัดที่ 4 สร้างวินโดวส์หลักที่เป็นอิสระจาก root ซึ่งว่า top บรรทัดที่ 5 กำหนด title ของ top เท่ากับ "Toplevel" บรรทัดที่ 6 สั่งวินโดวส์ใหม่ด้วยเมธอด deiconify() ผลลัพธ์แสดงดังรูปด้านบน

โปรแกรมตัวอย่างที่ 19.27 แสดงตัวอย่างการสร้างและใช้งาน Toplevel อีกด้วยหนึ่ง

**Program Example 19.27: Other Toplevel**

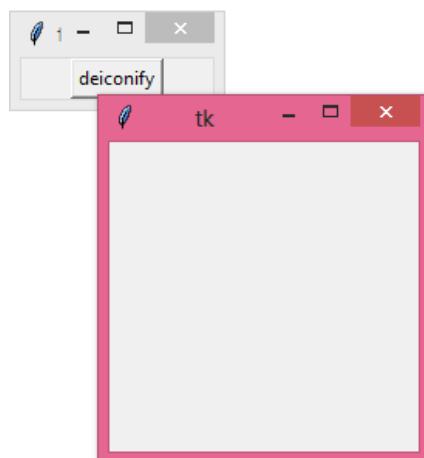
```
1 from tkinter import *
```

```

2
⇒3 root = Tk()
⇒4 root.withdraw()
5
⇒6 top = Toplevel(root)
⇒7 top.protocol("WM_DELETE_WINDOW", root.destroy)
8
⇒9 but = Button(top, text='deiconify')
⇒10 but['command'] = root.deiconify
11 but.pack()
12
13 root.mainloop()

```

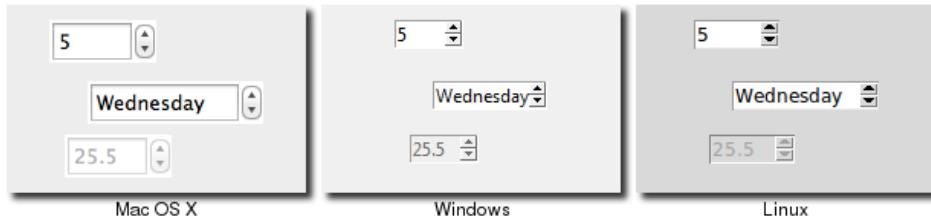
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



ตัวอย่างโปรแกรมที่ 19.27 บรรทัดที่ 3 สร้างหน้าต่างหลักชื่อ root จากนั้นบรรทัดที่ 4 โปรแกรมลบวินโดว์ด้วยเมธอด withdraw() บรรทัดที่ 6 โปรแกรมสร้างวินโดว์ใหม่ชื่อ top โดยสืบทอดคุณสมบัติทั้งหมดมาจากวินโดว์ root บรรทัดที่ 7 โปรแกรมทำการลงทะเบียนเมธอด root.destroy กับ "WM\_DELETE\_WINDOW" เพื่อใช้สำหรับลบหรือทำลายวินโดว์ออกจากจอมอนิเตอร์ บรรทัดที่ 9 สร้างปุ่มชื่อ but มีข้อความบนปุ่มคือ 'deiconify' บรรทัดที่ 10 เป็นการกำหนดว่าเมื่อคลิกปุ่มดังกล่าว โปรแกรมจะเรียกเมธอด root.deiconify เพื่อยุติการทำงานของวินโดว์ ผลลัพธ์แสดงดังรูปด้านบน

## 16. Spinbox

Spinbox คือ Widget ที่เหมือนกับ Entry แต่สามารถกำหนดขอบเขตของข้อมูลได้ ดังแสดงในรูปที่ 19.28



รูปที่ 19.28 แสดงรูปแบบของ Spinbox บนระบบปฏิบัติการต่างๆ

รูปแบบคำสั่งสำหรับการสร้าง Spinbox คือ

```
s = Spinbox(root, option=value, ...)
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ Spinbox

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, bg, bd, command, cursor, disabledbackground, disabledforeground, font, fg, justify, relief, repeatdelay, repeatinterval, state, textvariable, width, wrap, xscrollcommand สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

| Option | คำอธิบาย                                                                                |
|--------|-----------------------------------------------------------------------------------------|
| format | กำหนดรูปแบบของสตริงในการแสดงผล เช่น Spinbox(root, from_=0, to=10, format='%10.4f')      |
| from_  | กำหนดจำนวนบรรทัดเริ่มต้นของ spinbox เช่น Spinbox(root, from_=0, to=10, format='%10.4f') |
| to     | กำหนดจำนวนบรรทัดสุดท้ายของ spinbox เช่น Spinbox(root, from_=0, to=10, format='%10.4f')  |

Widget ชนิด Spinbox มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด delete(startindex [,endindex]) ลบตัวอักษรหรือข้อความใน Spinbox โดยระบุช่วงของข้อความ ถ้ากำหนดเฉพาะ startindex ข้อความจะถูกลบตั้งแต่ startindex ถึงตำแหน่งสิ้นสุดของข้อความ เช่น spin.delete(1.0) โดย 1 หมายถึงบรรทัดที่ 1, 0 หมายถึงอักษรตัวที่ 0 (*line number.character number*)
- เมธอด get(startindex [,endindex]) คืนค่าเป็นตัวอักษรหรือข้อความจาก Spinbox โดยใช้ startindex กำหนดตำแหน่งเริ่มต้น และตำแหน่งสุดท้ายของข้อความด้วย endindex เช่น print(spin.get(1.0, END))

- เมธอด `index(index)` คืนค่าสมบูรณ์ (absolute value) จาก Spinbox โดยใช้ `index` กำหนดตำแหน่ง เช่น `print(spin.index(4.0))`
- เมธอด `insert(index [,string]...)` แทรกข้อความในตำแหน่งที่ระบุใน `index` เช่น `spin.insert(1.0, "Python")`

สำหรับตัวอย่างการสร้างและใช้งาน Spinbox แสดงในโปรแกรมตัวอย่างที่ 19.28 ดังนี้

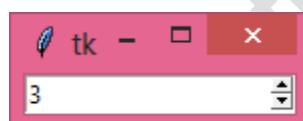
#### Program Example 19.28: Spinbox

```

1 from tkinter import *
2
3 root = Tk()
4
5 spin = Spinbox(root, from_=0, to=10)
6 spin.pack()
7
8 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรมที่ 19.28 แสดงการสร้างและใช้งาน Spinbox ในบรรทัดที่ 5 สร้าง Spinbox ชื่อ `spin` มีจำนวนบรรทัดเท่ากับ 10 บรรทัด เพื่อให้ผู้ใช้งานสามารถป้อนข้อมูลได้เพิ่มขึ้น โดยการคลิกเลือกที่ลูกศรขึ้นและลงที่อยู่ทางด้านขวาของ Spinbox

### 17. LabelFrame

LabelFrame คือ Widget ที่ผสมผสานกันระหว่าง Frame กับ Label นั่นคือ มีความสามารถในการรองรับ Widgets ต่างๆ เมื่อเฟรม และจัดการกับข้อความได้เหมือนกับ Label นั่นเองดังรูปที่ 19.29



รูปที่ 19.29 แสดงรูปแบบของ LabelFrame

รูปแบบคำสั่งสำหรับการสร้าง LabelFrame คือ

```
lf = LabelFrame(root, option=value, ...)
```

พารามิเตอร์คือ

- root คือ วินโดว์หลัก (root window)
- option คือ คุณสมบัติต่างๆ ของ LabelFrame

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, height, highlightbackground, highlightcolor, highlightthickness, relief, text, width สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

| Option      | คำอธิบาย                                                                                                                                                                                                                    |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| labelAnchor |  <p>กำหนดตำแหน่งที่จะวางเลbel ค่าดีฟอลต์คือ nw ซึ่งมีรูปแบบดังรูป เช่น LabelFrame(root, text="Group", labelanchor="n", padx=5, pady=5)</p> |

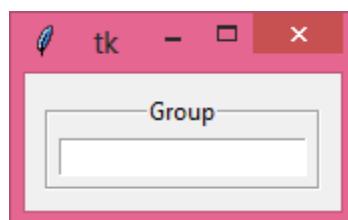
สำหรับตัวอย่างการสร้างและใช้งาน LabelFrame แสดงในโปรแกรมตัวอย่างที่ 19.29 ดังนี้

#### Program Example 19.29: LabelFrame

```

1 from tkinter import *
2
3 root = Tk()
4
5 lf = LabelFrame(root, text="Group", labelanchor="n", padx=5,
6 pady=5)
7 lf.pack(padx=10, pady=10)
8
9 e = Entry(lf)
10 e.pack()
11 root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง

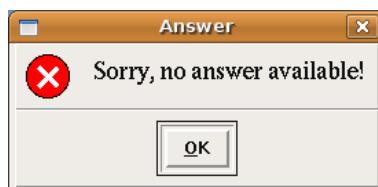


จากตัวอย่างโปรแกรมที่ 19.29 แสดงการสร้างและใช้งาน LabelFrame ในบรรทัดที่ 5 สร้าง

LabelFrame ชื่อ lf โดยมีข้อความว่า "Group" วางอยู่ในทิศเหนือ (labelanchor="n") ของวินโดว์บรรทัดที่ 8 สร้าง Entry และเพิ่มลงใน LabelFrame

## 18. MessageBox

MessageBox คือ Widget ที่ใช้สำหรับแสดงข้อความที่เหมาะสม หรือตามที่ผู้เขียนโปรแกรมต้องการ เช่น ข้อความเกี่ยวกับการกระทำที่ผิดพลาดของผู้ใช้งาน ข้อความแจ้งเตือน ต่างๆ เป็นต้น ดังรูปที่ 19.30



รูปที่ 19.30 แสดงรูปแบบของ MessageBox

รูปแบบคำสั่งสำหรับการสร้าง MessageBox คือ

```
mb = messagebox.FunctionName(title, message [, options])
```

พารามิเตอร์คือ

- title คือ title ของ messagebox
- message คือ ข้อความที่ต้องการแสดงใน messagebox
- options คือ คุณสมบัติต่างๆ ของ MessageBox เช่น ABORT, RETRY หรือ IGNORE

สำหรับฟังชัน (FunctionName) ที่สามารถใช้งานได้ดังนี้

- showinfo()
- showwarning()
- showerror()
- askquestion()
- askokcancel()
- askyesno()
- askretrycancel()

สำหรับตัวอย่างการสร้างและใช้งาน messagebox และแสดงในโปรแกรมตัวอย่างที่ 19.30 ดังนี้

### Program Example 19.30: messagebox

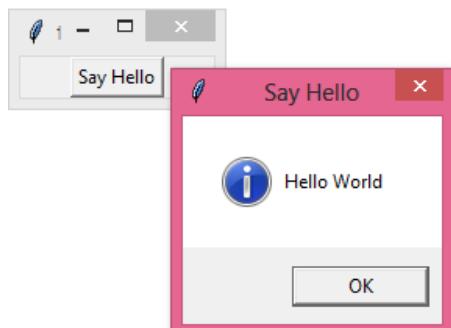
```
1 from tkinter import *
2 import tkinter
```

```

3
4 root = Tk()
5 ↪5 def hello():
6 messagebox.showinfo("Say Hello", "Hello World")
7
8 ↪8 B1 = Button(root, text = "Say Hello", command = hello)
9 B1.pack()
10
11 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



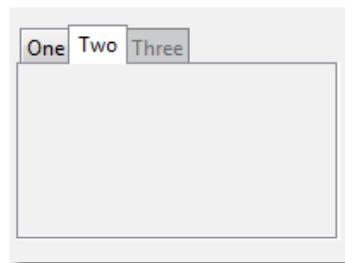
จากตัวอย่างโปรแกรมที่ 19.30 แสดงการสร้างและใช้งาน messagebox ในบรรทัดที่ 5 สร้างฟังชันชื่อ hello() โดยพิมพ์ข้อความว่า "Hello World" ผ่าน messagebox บรรทัดที่ 8 สร้างปุ่มชื่อ B1 มีข้อความว่า "Say Hello" เมื่อกดปุ่มดังกล่าว โปรแกรมจะเรียกฟังชัน hello() มาทำงาน

## 19. Widgets อื่น ๆ ที่น่าสนใจ

ไฟชอนยังมี Widgets ที่น่าสนใจอีก เช่น Paned Windows, Notebook, Tree, Combobox, SizeGrip, Progressbar ซึ่งมีรูปแบบคือ



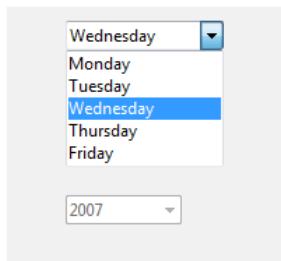
Paned Windows



Notebook

|           | Size  | Modified        |
|-----------|-------|-----------------|
| widgets   | 25KB  | Yesterday       |
| gallery   | 2KB   | Two weeks ago   |
| resources | 220KB | Three weeks ago |
| tutorial  | 21MB  | Ten minutes ago |
| canvas    | 18KB  | Last week       |
| tree      | 5KB   | Ten minutes ago |
| text      | 12KB  | Yesterday       |

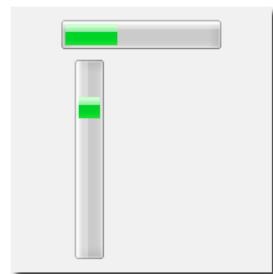
Tree



Combobox



SizeGrip



Progressbar

ตัวอย่างโปรแกรมที่ 19.31 แสดงการสร้างและใช้งาน Widgets ต่างๆ ที่แสดงไว้ในรูปด้านบน

**Program Example 19.31: Other Widgets**

```

1 from tkinter import *
2 import tkinter
3 from tkinter import ttk
4
5 root = Tk()
6
7 # PanedWindow
8 p = PanedWindow(root, orient=VERTICAL)
9 f1 = LabelFrame(p, text='Panel1', width=100, height=100)
10 f2 = LabelFrame(p, text='Panel2', width=100, height=100)
11 p.add(f1)
12 p.add(f2)
13 p.pack()
14
15 # Notebook
16 n = ttk.Notebook(root)
17 f1 = Frame(n)
18 f2 = Frame(n)
19 n.add(f1, text='One')
20 n.add(f2, text='Two')
21 n.pack()
22
23 # Treeview
24 tree = ttk.Treeview(root)
25 tree.insert('', 'end', 'widgets', text='Widget Tour')
26 tree.insert('', 0, 'gallery', text='Applications')
27 id = tree.insert('', 'end', text='Tutorial')
28 tree.insert('widgets', 'end', text='Canvas')
29 tree.insert(id, 'end', text='Tree')
30 tree.pack()
31
32 # Combobox
33 countryvar = StringVar()
34 country = ttk.Combobox(root, textvariable=countryvar)
35 country['values'] = ('USA', 'Canada', 'Australia')
36 country.pack()
37
38 # Sizegrip
39 sg = ttk.Sizegrip(root)

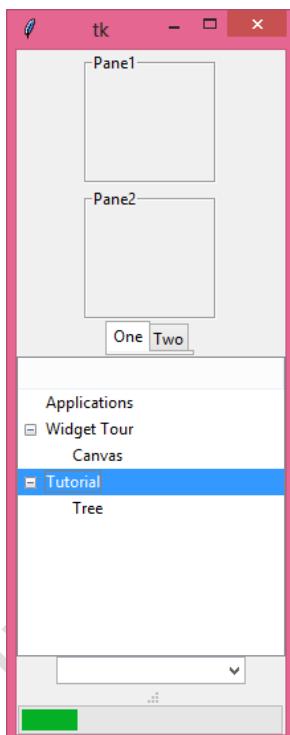
```

```

40 sg.pack()
41
⇒42 # Progressbar
43 p = ttk.Progressbar(root, orient=HORIZONTAL, length=200,
44 mode='determinate')
45 p.start()
46 p.pack()
47 root.mainloop()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



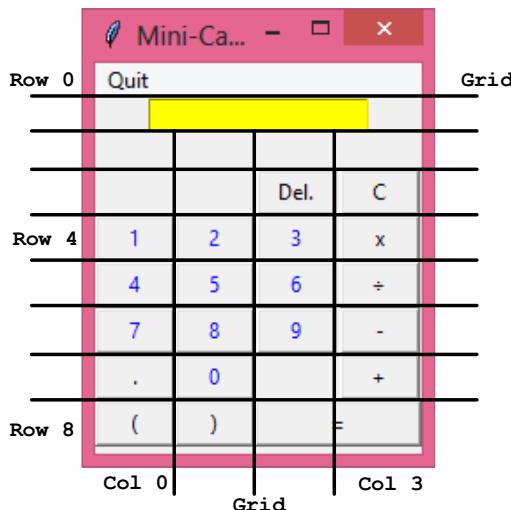
## 6. ตัวอย่างการประยุกต์ใช้งานไฟรอน GUI

ในหัวข้อนี้ผู้เขียนจะนำเสนอรูปแบบการใช้งานไฟรอน GUI ประยุกต์เข้ากับการใช้งานจริง ซึ่งมีเป้าหมายเพื่อต้องการให้ผู้อ่านเข้าใจวิธีการพัฒนาไฟรอน GUI ในภาพรวม โดยการนำเอา Widgets ต่างๆ ที่อธิบายมาแล้วในตอนต้น ประกอบเข้าเป็นแอพพลิเคชันที่มีความซับซ้อนมากขึ้น เพื่อเป็นแนวทางในการพัฒนาโปรแกรมต่อไปในอนาคตได้ ซึ่งจะแสดงไว้ 2 ตัวอย่างคือ โปรแกรมเครื่องคิดเลข และเกมส์ ทิก-แทค-โต ดังนี้

### 1. เครื่องคิดเลขขนาดเล็ก (Mini-Calculator)

การเขียนโปรแกรมเครื่องคิดเลขมีขั้นตอนในการออกแบบและพัฒนาโปรแกรมดังนี้

- 1) สเก็ตช์ (Sketch) เครื่องคิดเลขลงบนกระดาษแบบคร่าวๆ ดังรูปที่ 19.31



รูปที่ 19.31 แสดงภาพสเก็ตช์ของโปรแกรมเครื่องคิดเลข

2) นำเข้าโมดูลในการสร้าง GUI (tkinter)

```
from tkinter import *
```

3) ประกาศตัวแปรชนิดโภบล (global) เพื่อให้ฟังชันต่างๆ สามารถเรียกใช้งานได้ทุกๆ ที่ในโปรแกรม

```
expression = ""
```

4) สร้างหน้าต่างหลัก (วินโดว์ส์หลัก) หรือ root window

```
root = drawMainWindow()
```

5) สร้างเมนู (Menubar)

```
menu = drawMenuBar()
```

6) สร้างหน้าจอแสดงผลพื้นที่ของเครื่องคิดเลข (Entry Widget)

```
display = drawDisplay()
```

7) สร้างปุ่มของเครื่องคิดเลขทั้งหมด

```
drawCalButton()
```

8) เขียนโปรแกรมดักจับและควบคุมเหตุการณ์ต่างๆ ที่เกิดขึ้นบนเครื่องคิดเลข handleEvents()

9) กำหนดให้โปรแกรมวนลูบรับคำสั่งไปเรื่อยๆ จนกว่าจะยุติโปรแกรม

```
mainloop()
```

ตัวอย่างโปรแกรมชื่อ calculator.py แสดงตัวอย่างโปรแกรมเครื่องคิดเลขขนาดเล็ก

|                                                    |  |
|----------------------------------------------------|--|
| <b>Program Example Calculator: Mini-Calculator</b> |  |
|----------------------------------------------------|--|

|    |                       |
|----|-----------------------|
| ⇒1 | from tkinter import * |
|----|-----------------------|

|   |  |
|---|--|
| 2 |  |
|---|--|

|    |                 |
|----|-----------------|
| ⇒3 | expression = "" |
|----|-----------------|

```

4
⇒5 def drawMainWindow():
5 root = Tk()
6 root.title("Mini-Calculator")
7 return root
8
9
⇒10 def drawMenuBar(root):
10 menu = Menu(root)
11 menu.add_command(label="Quit", command=root.destroy)
12 root.config(menu=menu)
13 return menu
14
15
⇒16 def drawDisplay(root):
16 display = Entry(root, bg="yellow", fg="red")
17 display.grid(row=1, column=0, columnspan=5)
18 Label(root).grid(row=2, column=0)
19 return display
20
21
⇒22 def collectExpression(keypress):
22 global expression
23 expression += keypress
24 print(expression)
25 return expression
26
27
⇒28 def insertExpression(display, keypress):
28 display.insert(END, keypress)
29
30
⇒31 def clearDisplay(display, mode):
31 global expression
32 if mode == "DEL":
33 expression = expression[0:len(display.get())-1]
34 display.delete(len(display.get())-1, END)
35 print(expression)
36 elif mode == "DELALL":
37 expression = ""
38 display.delete(0, END)
39 print(expression)
40
41
⇒42 def handleEvents(display, keypress):
42 if keypress == "C":
43 clearDisplay(display, "DELALL")
44 elif keypress == "DEL":
45 clearDisplay(display, "DEL")
46 elif keypress == ".":
47 insertExpression(display, keypress)
48 collectExpression(keypress)
49 elif keypress == "(":
50 insertExpression(display, keypress)
51 collectExpression(keypress)
52 elif keypress == ")":
53 insertExpression(display, keypress)
54 collectExpression(keypress)
55 elif keypress == "+":
56 insertExpression(display, keypress)
57

```

```

58 collectExpression(keypress)
59 elif keypress == "-":
60 insertExpression(display, keypress)
61 collectExpression(keypress)
62 elif keypress == "x":
63 insertExpression(display, keypress)
64 collectExpression("*")
65 elif keypress == "÷":
66 insertExpression(display, keypress)
67 collectExpression("/")
68 elif keypress == "=":
69 global expression
70 try:
71 result = eval(expression)
72 print(result)
73 clearDisplay(display, "DELALL")
74 insertExpression(display, str(result))
75 except:
76 clearDisplay(display, "DELALL")
77 insertExpression(display, "Error: Can't
execute")
78 else:
79 insertExpression(display, keypress)
80 collectExpression(keypress)
81
⇒ 82 def drawCalButton(root, display):
83 Button(root, text="1", width=5, foreground="blue",
84 command=lambda: handleEvents(display, "1")).grid(row=4,
85 column=0)
86 Button(root, text="2", width=5, foreground="blue",
87 command=lambda: handleEvents(display, "2")).grid(row=4,
88 column=1)
89 Button(root, text="3", width=5, foreground="blue",
90 command=lambda: handleEvents(display, "3")).grid(row=4,
91 column=2)
92 Button(root, text="4", width=5, foreground="blue",
93 command=lambda: handleEvents(display, "4")).grid(row=5,
94 column=0)
95 Button(root, text="5", width=5, foreground="blue",
96 command=lambda: handleEvents(display, "5")).grid(row=5,
97 column=1)
98 Button(root, text="6", width=5, foreground="blue",
99 command=lambda: handleEvents(display, "6")).grid(row=5,
100 column=2)
101 Button(root, text="7", width=5, foreground="blue",
102 command=lambda: handleEvents(display, "7")).grid(row=6,
103 column=0)
104 Button(root, text="8", width=5, foreground="blue",
105 command=lambda: handleEvents(display, "8")).grid(row=6,
106 column=1)
107 Button(root, text="9", width=5, foreground="blue",
108 command=lambda: handleEvents(display, "9")).grid(row=6,
109 column=2)

```

```

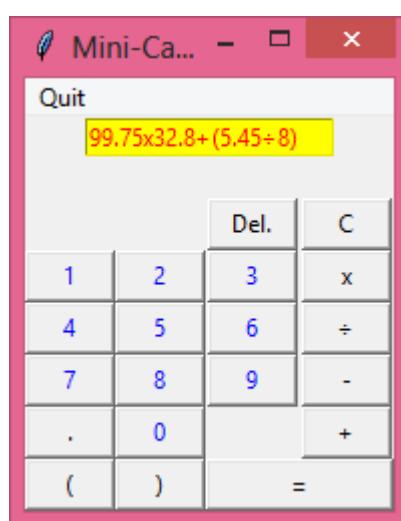
92 Button(root, text="0", width=5, foreground="blue",
93 command=lambda: handleEvents(display, "0")).grid(row=7,
94 column=1)
95 Button(root, text=". ", width=5, command=lambda:
96 handleEvents(display, ". ")).grid(row=7, column=0)
97 Button(root, text="= ", width=12, command=lambda:
98 handleEvents(display, "=")).grid(row=8, column=2,
99 columnspan=2)
100 Button(root, text="(", width=5, command=lambda:
101 handleEvents(display, "(")).grid(row=8, column=0)
102 Button(root, text=")", width=5, command=lambda:
103 handleEvents(display, ") ")).grid(row=8, column=1)
104 Button(root, text="x", width=5, command=lambda:
105 handleEvents(display, "x ")).grid(row=4, column=3)
106 Button(root, text="÷", width=5, command=lambda:
107 handleEvents(display, "÷ ")).grid(row=5, column=3)
108 Button(root, text="-", width=5, command=lambda:
109 handleEvents(display, "- ")).grid(row=6, column=3)
110 Button(root, text="+", width=5, command=lambda:
111 handleEvents(display, "+ ")).grid(row=7, column=3)
112 Button(root, text="C", width=5, command=lambda:
113 handleEvents(display, "C ")).grid(row=3, column=3)
114 Button(root, text="Del.", width=5, command=lambda:
115 handleEvents(display, "DEL ")).grid(row=3, column=2)

⇒104 def calculator():
105 root = drawMainWindow()
106 menu = drawMenuBar(root)
107 display = drawDisplay(root)
108 drawCalButton(root, display)
109 mainloop()

⇒111 if __name__ == '__main__':
112 calculator()

```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



จากตัวอย่างโปรแกรม calculator.py แสดงการสร้างเครื่องคิดเลขขนาดเล็ก ในที่นี้ผู้เขียนจะขอ อธิบายโปรแกรมเฉพาะส่วนที่สำคัญตามลำดับการพัฒนาโปรแกรมที่ได้กล่าวมาแล้วในเบื้องต้น โดยเริ่ม จากบรรทัดที่ 1 เป็นการนำเข้าโมดูล tkinter เพื่อสร้าง GUI สำหรับเครื่องคิดเลข บรรทัดที่ 3 ประกาศ ตัวแปรชนิดโภคอลชื่อ expression เพื่อใช้สำหรับเก็บนิพจน์คณิตศาสตร์ที่ผู้ใช้ป้อนให้กับเครื่องคิดเลข บรรทัดที่ 5 สร้างหน้าต่างหลัก โดยมี title คือ "Mini-Calculator" บรรทัดที่ 10 สร้างเมนูที่มีเมนูย่อย เพียงเมนูเดียวคือ Quit เพื่อใช้สำหรับออกจากโปรแกรม บรรทัดที่ 16 สร้าง Entry เพื่อรับข้อมูลจาก ผู้ใช้งานเพื่อคำนวณผลลัพธ์ในโปรแกรมเครื่องคิดเลข โดยวางลงบนกริดในตำแหน่งแรกที่ 1 คอลัมน์ที่ 0 และวางเลbelเพื่อคืนระหว่างปุ่มเครื่องคิดเลขกับ Entry โดยวางอยู่บนกริดในตำแหน่งแรก ที่ 2 คอลัมน์ที่ 0

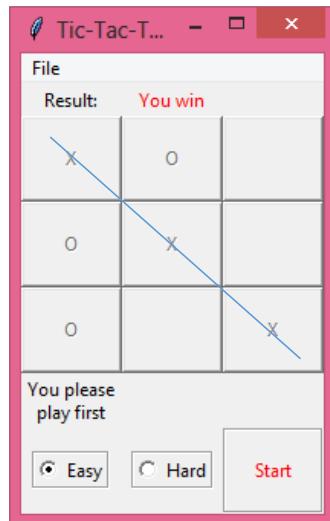
บรรทัดที่ 22 สร้างฟังชันชื่อ collectExpression() ทำหน้าที่เชื่อมต่อนิพจน์ของคณิตศาสตร์ที่ ผู้ใช้งานป้อนเข้ามาด้วยปุ่ม บรรทัดที่ 28 สร้างฟังชันชื่อ insertExpression() ทำหน้าที่แสดงนิพจน์คณิต ศาสตร์ออกทางจอภาพ บรรทัดที่ 31 สร้างฟังชันชื่อ clearDisplay() ทำหน้าที่ลบนิพจน์คณิตศาสตร์แบบ ครั้งเดียวทั้งหมด และลบแบบทีละตัว บรรทัดที่ 42 สร้างฟังชันชื่อ handleEvents() ทำหน้าที่ตอบสนอง เมื่อผู้ใช้งานกดปุ่มต่างๆ บนเครื่องคิดเลข บรรทัดที่ 82 สร้างฟังชันชื่อ drawCalButton() ทำหน้าที่วาด Widgets ต่างๆ ลงบนหน้าต่างวินโดว์ บรรทัดที่ 104 สร้างฟังชัน calculator() ทำหน้าที่ควบคุมการ ทำงานทั้งหมดของโปรแกรม โดยรวมฟังชันหลักๆ เข้าไว้ด้วยกัน บรรทัดที่ 111 โปรแกรม ตรวจสอบว่ามีการสร้างตัวแปร \_\_main\_\_ ไว้หรือไม่ (โดยปกติพythonจะสร้างตัวแปร \_\_main\_\_ ไว้ แล้วอัตโนมัติ) ถ้าประการตัวแปรดังกล่าวไว้ในโปรแกรม จะส่งผลให้สามารถเรียกฟังชัน calculator() เข้ามาทำงานได้ เพราะเงื่อนไขใน if เป็นจริงนั่นเอง



**Tips:** การใช้งาน option: command ของ Button โดยปกติจะเรียกใช้งานได้โดยตรง เช่น command=callback แต่โปรแกรมจะทำงานได้อย่างถูกต้อง เมื่อฟังชันดังกล่าวไม่มีการส่ง พารามิเตอร์ให้กับฟังชัน แต่ถ้าจำเป็นต้องส่งพารามิเตอร์ให้กับใช้ฟังชันควรใช้ lambda แทน เช่น lambda: handleEvents(display, "=")

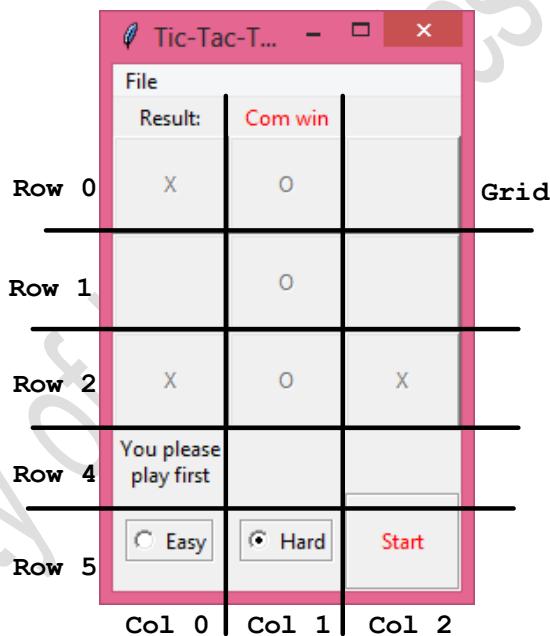
## 2. เกมส์ Tic-Tac-Toe

เกมส์ Tic-Tac-Toe หรือคนไทยรู้จักกันในชื่อ เกมส์ O (โอ) – X (เอ็กซ์) วิธีการเล่นก็ไม่ ยาก โดยผู้เล่นจะชนะได้ก็ต่อเมื่อ ผู้เล่นต้องเลือก X (เมื่อผู้เล่นเลือก X) ให้ครบ 3 ตัวต่อเนื่องกัน ไม่ว่าจะเป็นแนวอน แนวตั้ง หรือแนวทะแยงก์ได้ ดังรูปด้านล่าง



การเขียนโปรแกรมเกมส์ Tic-Tac-Toe มีขั้นตอนในการออกแบบและพัฒนาโปรแกรมดังนี้

- 1) สเก็ตช์ (Sketch) เกมส์ Tic-Tac-Toe ลงบนกระดาษแบบคร่าวๆ ดังรูปที่ 19.32



รูปที่ 19.32 แสดงภาพสเก็ตช์ของโปรแกรม Tic-Tac-Toe

- 2) นำเข้าโมดูลในการสร้าง GUI (tkinter) และโมดูลอื่นๆ ที่เกี่ยวข้อง
- 3) ประกาศตัวแปรชนิดโภบล (global) เพื่อให้ฟังชันต่างๆ สามารถเรียกใช้งานได้ทุกๆ ที่ในโปรแกรม
- 4) สร้างหน้าต่างหลัก (วินโดว์หลัก) หรือ root window
- 5) สร้างเมนู (Menubar)
- 6) สร้างหน้าจอแสดงผลลัพธ์ของการเล่นเกมส์ (Label Widget)
- 7) สร้างอินเตอร์เฟสสำหรับเล่นของเกมส์และปุ่มควบคุมทั้งหมด

- 8) เขียนโปรแกรมดักจับและควบคุมเหตุการณ์ต่างๆ ที่เกิดขึ้นบนเกมส์
- 9) กำหนดให้โปรแกรมวนลูบรับคำสั่งไปเรื่อยๆ จนกว่าจะยุติโปรแกรม

สำหรับตัวอย่างรหัสต้นฉบับ (Source Code) โปรแกรมเกมส์ Tic-Tac-Toe แสดงดังต่อไปนี้

**Program Example Tic-Tac-Toe:**

```

⇒1 from tkinter import *
2 import tkinter
3 import random
4
⇒5 MAP = [-1, -1, -1,
6 -1, -1, -1,
7 -1, -1, -1]
8
⇒9 layoutGrid = [(0, 1, 0), (1, 1, 1), (2, 1, 2),
10 (3, 2, 0), (4, 2, 1), (5, 2, 2),
11 (6, 3, 0), (7, 3, 1), (8, 3, 2)]
12
⇒13 root = Tk()
14 layoutButtons = [] #keeps all game buttons
15 controlButtons = [] #keeps all control buttons
16 player = 1 #0 = Computer, 1 = You (default=You)
17 playMode = 0 # 0 = Easy, 1 = Hard
18
⇒19 def drawMainWindow():
20 global root
21 root.title("Tic-Tac-Toe")
22
⇒23 def drawMenuBar():
24 menubar = Menu(root, selectcolor="red")
25 filemenu = Menu(menubar, tearoff=0)
26 filemenu.add_command(label="New Game",
command=resetGame)
27 filemenu.add_separator()
28 filemenu.add_command(label="Exit", command=root.destroy)
29 menubar.add_cascade(label="File", menu=filemenu)
30 root.config(menu=menubar)
31 return menubar
32
⇒33 def setMode(mode):
34 def set():
35 global playMode
36 if mode == 0: playMode = 0
37 else: playMode = 1
38 return set
39
⇒40 def resetGame():
41 global MAP
42 global layoutButtons
43 global controlButtons
44 controlButtons[1]['text']=""
45 layoutButtons = []
46 controlButtons = []

```

```

47 playMode = 0
48 for i in range(9):
49 MAP[i] = -1
50 TicTacToe()
51
⇒52 def checkStatus():
53 global MAP
54 for i in range(9):
55 if MAP[i] == -1:
56 return True
57 return False
58
⇒59 def comPlayModeEasy():
59 global MAP
60 global layoutButtons
61 while checkStatus():
62 ran = random.randrange(9)
63 if MAP[ran] == -1:
64 MAP[ran] = 0
65 layoutButtons[ran]['text'] ='O'
66 layoutButtons[ran]['state']=DISABLED
67 break
68
69
⇒70 def comPlayModeAI():
70 global MAP
71 global layoutButtons
72 while checkStatus():
73 # AI computing
74 searchCom = AICheck(0)
75 searchPly = AICheck(1)
76 if searchCom == -1 and searchPly == -1:
77 ran = random.randrange(9)
78 if MAP[ran] == -1:
79 MAP[ran] = 0
80 layoutButtons[ran]['text'] ='O'
81 layoutButtons[ran]['state']=DISABLED
82 break
83
84 else:
85 if searchCom != -1:
86 MAP[searchCom] = 0
87 layoutButtons[searchCom]['text'] ='O'
88 layoutButtons[searchCom]['state']=DISABLED
89 break
90 else:
91 MAP[searchPly] = 0
92 layoutButtons[searchPly]['text'] ='O'
93 layoutButtons[searchPly]['state']=DISABLED
94 break
95
⇒96 def AICheck(who):
96 global MAP
97 count = 0
98 for i in 0,1,2:
99 if MAP[i]==who:
100 count += 1

```

```
102 if count == 2:
103 for j in 0,1,2:
104 if MAP[j]== -1:
105 return j
106 count = 0
107 for i in 3,4,5:
108 if MAP[i]==who:
109 count += 1
110 if count == 2:
111 for j in 3,4,5:
112 if MAP[j]== -1:
113 return j
114 count = 0
115 for i in 6,7,8:
116 if MAP[i]==who:
117 count += 1
118 if count == 2:
119 for j in 6,7,8:
120 if MAP[j]== -1:
121 return j
122 count = 0
123 for i in 0,3,6:
124 if MAP[i]==who:
125 count += 1
126 if count == 2:
127 for j in 0,3,6:
128 if MAP[j]== -1:
129 return j
130 count = 0
131 for i in 1,4,7:
132 if MAP[i]==who:
133 count += 1
134 if count == 2:
135 for j in 1,4,7:
136 if MAP[j]== -1:
137 return j
138 count = 0
139 for i in 2,5,8:
140 if MAP[i]==who:
141 count += 1
142 if count == 2:
143 for j in 2,5,8:
144 if MAP[j]== -1:
145 return j
146 count = 0
147 for i in 0,4,8:
148 if MAP[i]==who:
149 count += 1
150 if count == 2:
151 for j in 0,4,8:
152 if MAP[j]== -1:
153 return j
154 count = 0
155 for i in 2,4,6:
156 if MAP[i]==who:
```

```

157 count += 1
158 if count == 2:
159 for j in 2,4,6:
160 if MAP[j]== -1:
161 return j
162 return -1
163
⇒164 def checkWin():
165 global MAP
166 for i in range(2):
167 if MAP[0]==i and MAP[1]==i and MAP[2]==i: #(r0)
168 if i == 0: return "COM"
169 elif i == 1: return "YOU"
170 elif MAP[3]==i and MAP[4]==i and MAP[5]==i: #(r1)
171 if i == 0: return "COM"
172 elif i == 1: return "YOU"
173 elif MAP[6]==i and MAP[7]==i and MAP[8]==i: #(r2)
174 if i == 0: return "COM"
175 elif i == 1: return "YOU"
176 elif MAP[0]==i and MAP[3]==i and MAP[6]==i: #(c0)
177 if i == 0: return "COM"
178 elif i == 1: return "YOU"
179 elif MAP[1]==i and MAP[4]==i and MAP[7]==i: #(c1)
180 if i == 0: return "COM"
181 elif i == 1: return "YOU"
182 elif MAP[2]==i and MAP[5]==i and MAP[8]==i: #(c2)
183 if i == 0: return "COM"
184 elif i == 1: return "YOU"
185 elif MAP[0]==i and MAP[4]==i and MAP[8]==i: #(s1)
186 if i == 0: return "COM"
187 elif i == 1: return "YOU"
188 elif MAP[2]==i and MAP[4]==i and MAP[6]==i: #(s1)
189 if i == 0: return "COM"
190 elif i == 1: return "YOU"
191
⇒192 def onClick(userMarks):
192 def click():
193 global MAP
194 global playMode
195 global layoutButtons
196 print(playMode)
197 if playMode == 0:
198 MAP[userMarks] = 1
199 layoutButtons[userMarks]['text']='X'
200 layoutButtons[userMarks]['state']=DISABLED
201 comPlayModeEasy()
202 check = checkWin()
203 if check == "YOU":
204 for button in layoutButtons:
205 button['state']=DISABLED
206 controlButtons[1]['fg']="red"
207 controlButtons[1]['text']="You win"
208
209 if check == "COM":
210 for button in layoutButtons:
211

```

```

212 button['state']=DISABLED
213 controlButtons[1]['fg']="red"
214 controlButtons[1]['text']="Com win"
215
216 if playMode == 1:
217 MAP[userMarks] = 1
218 layoutButtons[userMarks]['text']='X'
219 layoutButtons[userMarks]['state']=DISABLED
220 comPlayModeAI()
221 print(MAP)
222 check = checkWin()
223 if check == "YOU":
224 for button in layoutButtons:
225 button['state']=DISABLED
226 controlButtons[1]['fg']="red"
227 controlButtons[1]['text']="You win"
228
229 if check == "COM":
230 for button in layoutButtons:
231 button['state']=DISABLED
232 controlButtons[1]['fg']="red"
233 controlButtons[1]['text']="Com win"
234
235 return click
236
⇒236 def drawGraphic():
237 global root
238 global layoutButtons
239 global layoutGrid
240 global controlButtons
241 control = Label(root, text="Result:")
242 control.grid(row=0, column=0)
243 controlButtons.append(control) #controlButtons[0]
244 control = Label(root, text="")
245 control.grid(row=0, column=1)
246 controlButtons.append(control) #controlButtons[1]
247 control = Label(root, text="You please\nplay first")
248 control.grid(row=4, column=0)
249 controlButtons.append(control) #controlButtons[2]
250 control = Label(root, text="")
251 control.grid(row=4, column=1)
252 controlButtons.append(control) #controlButtons[3]
253 control = Radiobutton(root, text="Easy", relief=GROOVE,
254 value=0, command=setMode(0))
255 control.grid(row=5, column=0)
256 controlButtons.append(control) #controlButtons[4]
257 control = Radiobutton(root, text="Hard", relief=GROOVE,
258 value=1, command=setMode(1))
259 control.grid(row=5, column=1)
260 controlButtons.append(control) #controlButtons[5]
261 control = Button(root, text="Start", width=8, height=3,
262 relief=GROOVE, foreground="red", command=resetGame)
263 control.grid(row=5, column=2)
264 controlButtons.append(control) #controlButtons[6]
265
266 for i,j in enumerate(layoutGrid):

```

```

264 button = Button(root, text="", width=8, height=3,
265 foreground="blue", command=onClick(j[0]))
266 button.grid(row=j[1], column=j[2])
267 layoutButtons.append(button)
268
⇒268 def TicTacToe():
269 drawMainWindow()
270 drawMenuBar()
271 drawGraphic()
272 mainloop()
273
⇒274 if __name__ == '__main__':
⇒275 TicTacToe()

```

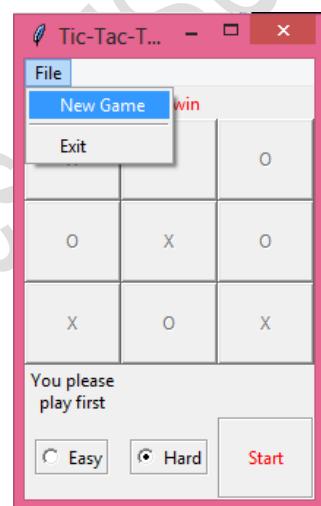
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูปด้านล่าง



เริ่มต้นเล่นเกมส์



ผู้เล่นเกมส์ชนะ



คลิกเลือกเมนูบนเกมส์

จากตัวอย่างโปรแกรม Tic-Tac-Toe บรรทัดที่ 1 แสดงการนำเข้าโมดูล tkinter และ random เพื่อใช้สำหรับสร้างกราฟฟิกและสุ่มค่าให้กับโปรแกรม บรรทัดที่ 5 ประกาศตัวแปรโกลบอลชื่อ MAP ทำหน้าที่กำหนดค่าเริ่มต้นให้กับแผนที่ของเกมส์ โดยค่าเริ่มต้นเป็น -1 ถ้าผู้ใช้คลิกเล่นเกมส์ โปรแกรมจะกำหนดค่าให้กับตำแหน่งบน MAP เป็น 1 แต่ถ้าคอมพิวเตอร์เล่นจะกำหนดเป็น 0 ตารางตั้งกล่าวจะใช้เพื่อตรวจสอบว่าใครเป็นผู้ชนะระหว่างผู้เล่นหรือคอมพิวเตอร์ รวมถึงคอมพิวเตอร์จะใช้ตารางดังกล่าวตรวจสอบว่าควรจะคลิกเลือกช่องไหน (สำหรับหมวด AI คอมพิวเตอร์จะต้องฉลาดที่จะคลิกเลือก) บรรทัดที่ 9 ประกาศตัวแปรชื่อ layoutGrid ทำหน้าที่กำหนดตำแหน่งการวางปุ่มต่างๆ วงบวนโดยส์ บรรทัดที่ 13 – 17 ประกาศตัวแปรชนิดโกลบอล ประกอบไปด้วยหน้าต่างหลัก (root), ตัวแปรชนิดลิสต์ที่ใช้สำหรับเก็บข้อมูลอปเจกต์ปุ่มเล่นเกมส์ (layoutButtons), ตัวแปรชนิดลิสต์ที่ใช้สำหรับเก็บข้อมูลอปเจกต์ปุ่มสำหรับควบคุมเกมส์ (controlButtons), ตัวแปรเก็บสถานะผู้เล่นคือ player ถ้าตัวแปรดังกล่าวมีค่าเท่ากับ 0 แสดงว่าเป็นคอมพิวเตอร์ ในทางกลับถ้าเป็น 1 แสดงว่าเป็นมนุษย์, ตัวแปรชื่อ playMode ทำหน้าที่เก็บโหมดการเล่นคือ ง่ายหรือยาก (โดยผู้ใช้คลิกเลือกที่ปุ่ม Radiobutton) ถ้าเป็น

โหนดง่ายโปรแกรมจะให้คอมพิวเตอร์สู่ตำแหน่งในการเล่นเกมส์ แต่ถ้าเป็นโหนดยาก โปรแกรมจะใช้ AI (Artificial Intelligence) ในการเลือกตำแหน่งในการเล่นเกมส์แทน

บรรทัดที่ 19 พังชันชื่อ drawMainWindow() ทำหน้าที่วาดหน้าต่างหลักเพื่อรับ Widgets ต่างๆ ที่จะวาดเป็น GUI ของเกมส์ บรรทัดที่ 23 พังชันชื่อ drawMenuBar() ทำหน้าที่วางแผน เมนู เช่น New Game และ Exit ลงบนหน้าต่างหลัก บรรทัดที่ 33 พังชันชื่อ setMode() ทำหน้าที่กำหนดโหมดการเล่น เกมส์ว่าง่ายหรือยาก (เมื่อดันถูกเรียกเมื่อกดปุ่ม radiobutton ของเกมส์) บรรทัดที่ 40 พังชันชื่อ resetGame() ทำหน้าที่วาดกราฟิกบนเกมส์ใหม่ทั้งหมด พร้อมเคลียร์ค่าตัวแปรต่างๆ สำหรับการเริ่มเล่นเกมส์ใหม่อีกครั้ง บรรทัดที่ 52 พังชันชื่อ checkStatus() ทำหน้าที่ตรวจสอบว่าข้อมูลใน MAP ยังมีตำแหน่งที่จะสามารถเล่นเกมส์ได้หรือไม่ ถ้าซองในตาราง MAP ไม่มีค่าเท่ากับ -1 แม้แต่ซองเดียว แสดงว่าไม่สามารถเล่นเกมส์ได้อีกแล้ว โปรแกรมจะคืนค่าเป็น False แต่ถ้ายังมีตำแหน่งในเกมส์ที่สามารถเล่นเกมส์ได้จะคืนค่าเป็น True

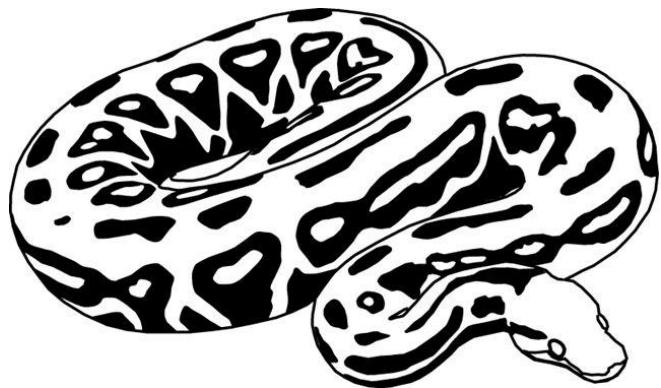
บรรทัดที่ 59 พังชันชื่อ comPlayModeEasy() ทำหน้าที่ค้นหาตำแหน่งซึ่งว่างที่คอมพิวเตอร์สามารถเล่นเกมส์ได้ใน MAP โดยใช้หลักการของการสุมค่า ถ้าค่าที่สูม (ตำแหน่งในตาราง MAP) ถูกเลือกไว้แล้ว โปรแกรมจะสุมไปเรื่อยๆ จนกว่าคอมพิวเตอร์จะสามารถวางตัวอักษร 'O' ได้สำเร็จ บรรทัดที่ 70 ในการนี้ที่ผู้เล่นเลือกเล่นแบบยาก (Hard) โปรแกรมจะเรียกใช้พังชันชื่อ comPlayModeAI() ซึ่งพังชันดังกล่าวจะคำนวณหาตำแหน่งที่คิดว่าได้เปรียบผู้เล่นมากที่สุด โดยอาศัยหลักการคือ ถ้าคอมพิวเตอร์วาง 'O' แล้วส่งผลให้ได้ตัวอักษร 'O' ครบ 3 ตัว ในแนวนอน แนวตั้ง หรือแนวทะแยง โปรแกรมจะเลือกตำแหน่งดังกล่าวทันที แต่ถ้าคอมพิวเตอร์ยังไม่มีแนวโน้มที่จะชนะผู้เล่นได้ คอมพิวเตอร์จะเลือกตำแหน่งที่ค้อยกีดกันไม่ให้ผู้เล่นวางอักษร 'X' ในตำแหน่งที่ผู้เล่นจะชนะคอมพิวเตอร์ได้ แต่ถ้าไม่เป็นไปตามที่กล่าวมาแล้วทั้ง 2 กรณี คอมพิวเตอร์จะหาตำแหน่งโดยการสุมแทน บรรทัดที่ 96 พังชันชื่อ AICheck() ทำหน้าที่ตรวจสอบข้อมูลในตาราง MAP ถึงความเป็นไปได้ที่จะชนะผู้เล่นในทุกกรณี และตำแหน่งที่จะขัดขวางไม่ให้ผู้เล่นชนะ ค่าที่คืนกลับจะเป็นตำแหน่งที่ได้เปรียบคู่แข่ง แต่ถ้าคอมพิวเตอร์ยังไม่มีแนวโน้มที่จะชนะจะคืนค่าเป็น -1

บรรทัดที่ 164 พังชันชื่อ checkWin() ทำหน้าที่ตรวจสอบว่าใครเป็นผู้ชนะระหว่างคอมพิวเตอร์ กับผู้เล่น โดยใช้หลักการตรวจสอบค่า 0 ('O' ในกรณีเป็นคอมพิวเตอร์) ในตาราง MAP ว่าเรียงตัวในแนวตั้ง แนวนอน หรือแนวทะแยงครบ 3 ค่าหรือไม่ สำหรับในกรณีตรวจสอบผู้เล่นจะเปลี่ยนเป็นตรวจสอบค่า 1 ('X' ในกรณีเป็นผู้เล่น) ในตาราง MAP แทน บรรทัดที่ 192 พังชันชื่อ onClick() ทำหน้าที่ดักจับหรือตอบสนองต่อผู้ใช้ เมื่อคลิกปุ่มต่างๆ บนเกมส์ บรรทัดที่ 236 พังชันชื่อ drawGraphic() ทำหน้าที่วาดปุ่มต่างๆ ของเกมส์ลงบนหน้าต่างหลัก บรรทัดที่ 268 พังชันชื่อ TicTacToe() ทำหน้าที่ควบคุมการสร้างเกมส์ทั้งหมดไว้ โดยเริ่มตั้งแต่การวาดหน้าต่างหลัก (drawMainWindow), การวาดเมนู

(drawMenuBar), การวาดกราฟฟิก (drawGraphic), และการวนลูบเพื่อให้เกมส์เล่นไปเรื่อยๆ อย่างต่อเนื่อง (mainloop) บรรทัดที่ 274 เป็นการตรวจสอบว่าตัวแปร \_\_main\_\_ มีการประกาศไว้หรือไม่ โดยปกติพชอนจะประกาศไว้เสมอทุกๆ โปรแกรมที่เขียนขึ้น ถ้ามีการประกาศตัวแปรดังกล่าวไว้ โปรแกรมจะเรียกฟังชัน TicTacToe() ขึ้นมาทำงานแทนที่

จบบทที่ 19

## จบภาค 4



## ภาคที่ 5

ไลบรารีพื้นฐานของไพธอน

 **python**  
Standard library of

พัฒนาภาษาในสำหรับข้อมูลพื้นฐาน

พัฒนาภาษาในสำหรับข้อมูลเชิงประวัติ

เมื่อต้องเกี่ยวกับการบริหารจัดการแฟ้มข้อมูลและไดเรคทรอรี่

## 1. พังชันภายในสำหรับข้อมูลพื้นฐาน (Built-in functions for basic data types)

พังชันภายใน (Built-in function) คือ พังชันที่ผู้เขียนโปรแกรมสามารถเรียกใช้งานได้ทันที โดยส่วนใหญ่ไม่ต้องนำเข้า (Import) มาจากไลบรารี (Library) แต่มีบางพังชันเท่านั้นที่จำเป็นต้องนำเข้ามาใช้งาน ซึ่งมีรูปแบบคือ

ตัวแปร.ชื่อพังชัน ([input\_var1, input\_var2, ..., input\_varN]) เช่น

```
str = "this is string example....wow!!!";
print ("str.capitalize() : ", str.capitalize())
```



OUTPUT

```
str.capitalize() : This is string example....wow!!!
```

ไฟรอน เป็นภาษาลูกผสมระหว่างภาษาเก่าและใหม่ ดังนั้นในไลบรารีของไฟรอนจะผสมกันระหว่างพังชัน และเมธอด (ใช้กับการเขียนโปรแกรมเชิงวัตถุ) ข้อสังเกตระหว่างพังชัน และเมธอดคือพังชันจะเรียกใช้งานโดยไม่จำเป็นต้องอ้างคลาสก่อน สามารถเรียกใช้งานได้โดยตรงดังนี้

`len(list)` คือการเรียกพังชัน

`list.count(Object)` คือการเรียกเมธอด โดยอ้าง `instance` ของคลาส โดยใช้ . เสมอ

## การเปลี่ยนชนิดข้อมูล (Data type conversion)

- `int()` แปลงข้อมูลจากชนิดต่างๆ เป็นเลขฐานสิบ

รูปแบบคำสั่ง: `int(x [,base])`

พารามิเตอร์: `x` คือข้อมูลชนิดใดๆ และ `base` คือเลขฐานสิบ สัญลักษณ์ [ ] หมายถึง `option` คือ กำหนด หรือไม่กำหนดก็ได้ ถ้าไม่กำหนดจะเป็นเลขฐานสิบโดยอัตโนมัติ

ค่าที่ส่งกลับ: จำนวนเต็มฐานสิบ

ตัวอย่าง:

```
num = int('10') # Convert string to integer
print(num)
print(type(num))
print(int('10',10)) # Define the 10 base
print(int(10.5)) # Convert float to integer
print(int(True)) # Convert Boolean to integer
z = 0b1001001; j = 0o35; k = 0x45A
print(int(z))
```

```
print(int(j))
print(int(k))
```

 **OUTPUT**

```
10
<class 'int'>
10
10
1
73
29
1114
```

- **float()** แปลงข้อมูลจากชนิดใดๆ เป็นเลขจำนวนจริงที่ต้องการ

รูปแบบคำสั่ง: float(x)

พารามิเตอร์: x คือข้อมูลชนิดใดๆ

ค่าที่ส่งกลับ: จำนวนจริง

ตัวอย่าง:

```
print(float(5))
print(float('5.35'))
print(float(False))
```

 **OUTPUT**

```
5.0
5.35
0.0
```

- **complex()** สร้างข้อมูลเชิงซ้อน

รูปแบบคำสั่ง: complex(real [,imag])

พารามิเตอร์: real คือข้อมูลบนแกนจริง และ imag คือข้อมูลบนแกนจินตภาพ

ค่าที่ส่งกลับ: จำนวนเชิงซ้อน

ตัวอย่าง:

```
complex_num = complex(5, 3)
print(complex_num)
complex_num = complex(5, -3)
print(complex_num)
```

 **OUTPUT**

```
(5+3j)
(5-3j)
```

- **str(x)** สร้างข้อมูลสตริง หรือสายอักขระ

รูปแบบคำสั่ง: str(x)

พารามิเตอร์: x คือ ข้อความเจ็กซ์เต๊ต

ค่าที่ส่งกลับ: ข้อมูลสตริง

ตัวอย่าง:

```
s = str(123)
print(s)
```

```
s = str(3.5)
print(s)
```



123  
3.5

**OUTPUT**

- **eval()** ทำหน้าที่ประเมินผล สร้างได้ ให้เป็นนิพจน์ทางคณิตศาสตร์ โดยปกติสร้างไม่สามารถประมาณผลได้ แต่ฟังชันนี้จะทำให้สร้างเป็นนิพจน์ทางคณิตศาสตร์ และสามารถประมาณได้

รูปแบบคำสั่ง: eval(str)

พารามิเตอร์: str คือ สริงที่ต้องการประเมินผล

ค่าที่ส่งกลับ: อ็อปเจกต์ได้

ตัวอย่าง:

```
s = '1.0 + 2.0'
print(s)
print(eval(s))
print(eval("'*' * 10"))
```



1.0 + 2.0  
3.0  
\*\*\*\*\*

**OUTPUT**

จากตัวอย่าง โดยปกติถ้าพิมพ์ '1.0 + 2.0' ด้วยคำสั่ง print ออกทางจอภาพ ผลที่ได้จะแสดงเป็น สริงคือ '1.0 + 2.0' แต่ถ้าผู้เขียนโปรแกรมต้องการให้เกิดการประมาณนิพจน์ดังกล่าว ให้ใช้ฟังชัน eval แทน print

- **tuple()** แปลงข้อมูลสริงให้เป็นข้อมูลชนิดทัพเพิล (tuple)

รูปแบบคำสั่ง: tuple(s)

พารามิเตอร์: s คือ สริงที่ต้องการแปลง

ค่าที่ส่งกลับ: ข้อมูลชนิดทัพเพิล

ตัวอย่าง:

```
s = '12345'
print(tuple(s))
```



('1', '2', '3', '4', '5')

**OUTPUT**

- **list()** แปลงข้อมูลชนิดสริงเป็นข้อมูลชนิดลิสต์ (list)

รูปแบบคำสั่ง: list(s)

พารามิเตอร์: s คือ สริงที่ต้องการแปลง

ค่าที่ส่งกลับ: ข้อมูลชนิดลิสต์

ตัวอย่าง:

```
s = '12345'
print(list(s)))
```



**OUTPUT**

```
['1', '2', '3', '4', '5']
```

- **set()** แปลงข้อมูลชนิดสตริงให้เป็นข้อมูลชนิดเซ็ต (Set)

รูปแบบคำสั่ง: set(s)

พารามิเตอร์: s คือ สตริงที่ต้องการแปลง

ค่าที่ส่งกลับ: ข้อมูลชนิดเซ็ต

ตัวอย่าง:

```
s = '12345'
print(set(s))
```



**OUTPUT**

```
{'1', '2', '3', '4', '5'}
```

- **dict()** สร้างข้อมูลชนิดดิกชันนารี (Dictionary) ซึ่งเป็นข้อมูลที่ประกอบด้วยคู่ของคีย์ (Key)

และค่าข้อมูล (Value) ที่อยู่ในรูปแบบ Key:Value

รูปแบบคำสั่ง: dict(d)

พารามิเตอร์: d คือ คู่ของข้อมูลที่ต้องการแปลง เช่น 001:"John" (Key=1, Value="John")

ค่าที่ส่งกลับ: ข้อมูลชนิดดิกชันนารี

ตัวอย่าง:

```
print(dict({"one":1,"two":2,"three":3}))
print(dict({1:'John',2:'Somsri',3:'Suchart'}))
```



**OUTPUT**

```
{'two': 2, 'three': 3, 'one': 1}
{1: 'John', 2: 'Somsri', 3: 'Suchart'}
```

- **hex()** แปลงเลขฐานได้ 4 เป็นเลขฐานสิบหก

รูปแบบคำสั่ง: hex(x)

พารามิเตอร์: x คือเลขฐานได้ 4 ที่ต้องการแปลง

ค่าที่ส่งกลับ: เลขฐานสิบหก

ตัวอย่าง:

```
x = 31; y = -15; z = 0b1001001; j = 0o35
print(hex(x))
print(hex(y))
print(hex(z))
```

```
print(hex(j))
```

|                                                                                   |                                               |
|-----------------------------------------------------------------------------------|-----------------------------------------------|
|  | 0x1f<br>-0xf<br><b>OUTPUT</b><br>0x49<br>0x1d |
|-----------------------------------------------------------------------------------|-----------------------------------------------|

- **oct(x)** แปลงเลขฐานไดๆ เป็นเลขฐานแปด

รูปแบบคำสั่ง: oct(x)

พารามิเตอร์: x คือเลขฐานไดๆ ที่ต้องการแปลง

ค่าที่ส่งกลับ: เลขฐานแปด

ตัวอย่าง:

```
x = 31; y = -15; z = 0b1001001; k = 0x45A
print(oct(x))
print(oct(y))
print(oct(z))
print(oct(k))
```

|                                                                                   |                                                   |
|-----------------------------------------------------------------------------------|---------------------------------------------------|
|  | 0o37<br>-0o17<br><b>OUTPUT</b><br>0o111<br>0o2132 |
|-----------------------------------------------------------------------------------|---------------------------------------------------|

## ฟังชันคำนวณทางคณิตศาสตร์ (Mathematical Functions)

- **abs(), fabs()** หากำจำนวนเต็มบวก (absolute) ของเลขจำนวนเต็มใดๆ

รูปแบบคำสั่ง: abs(x), fabs(x)

พารามิเตอร์: x คือจำนวนเต็มใดๆ

ค่าที่ส่งกลับ: เลขจำนวนเต็มบวก

ตัวอย่าง:

```
print ("abs(-45) : ", abs(-45))
print ("abs(100.12) : ", abs(100.12))
```

|                                                                                     |                                                        |
|-------------------------------------------------------------------------------------|--------------------------------------------------------|
|  | abs(-45) : 45<br>abs(100.12) : 100.12<br><b>OUTPUT</b> |
|-------------------------------------------------------------------------------------|--------------------------------------------------------|

- **ceil()** ปัดเศษขึ้นให้เป็นจำนวนเต็มที่ใกล้เคียงที่สุด

รูปแบบคำสั่ง: ceil(x)

พารามิเตอร์: x คือจำนวนเต็ม หรือจำนวนจริง

ค่าที่ส่งกลับ: เลขจำนวนเต็มที่ปัดเศษแล้ว

ตัวอย่าง:

```
import math # This will import math module
print ("math.ceil(-45.17) : ", math.ceil(-45.17))
```

```
print ("math.ceil(100.12) : ", math.ceil(100.12))
print ("math.ceil(100.72) : ", math.ceil(100.72))
print ("math.ceil(math.pi) : ", math.ceil(math.pi))
```



**OUTPUT**

```
math.ceil(-45.17) : -45
math.ceil(100.12) : 101
math.ceil(100.72) : 101
math.ceil(math.pi) : 4
```

- **exp()** คำนวณเลขชี้กำลัง

รูปแบบคำสั่ง: `exp(x)`

พารามิเตอร์:  $x$  คือจำนวนเต็ม หรือจำนวนจริง

ค่าที่ส่งกลับ: เลขยกกำลัง

ตัวอย่าง:

```
import math # This will import math module
print ("math.exp(-45.17) : ", math.exp(-45.17))
print ("math.exp(100.12) : ", math.exp(100.12))
print ("math.exp(100.72) : ", math.exp(100.72))
print ("math.exp(math.pi) : ", math.exp(math.pi))
```



**OUTPUT**

```
math.exp(-45.17) : 2.4150062132629406e-20
math.exp(100.12) : 3.0308436140742566e+43
math.exp(100.72) : 5.522557130248187e+43
math.exp(math.pi) : 23.140692632779267
```

- **floor()** คำนวณค่าจำนวนเต็มมากที่สุด แต่น้อยกว่าค่า  $x$

รูปแบบคำสั่ง: `floor(x)`

พารามิเตอร์:  $x$  คือจำนวนจริง

ค่าที่ส่งกลับ: เลขจำนวนเต็ม

ตัวอย่าง:

```
import math # This will import math module
print ("math.floor(-45.17):", math.floor(-45.17))
print ("math.floor(100.12):", math.floor(100.12))
print ("math.floor(100.72):", math.floor(100.72))
print ("math.floor(math.pi):", math.floor(math.pi))
```



**OUTPUT**

```
math.floor(-45.17) : -46
math.floor(100.12) : 100
math.floor(100.72) : 100
math.floor(math.pi) : 3
```

- **log()** คำนวณหาค่าลอการิทึม (Logarithm) ฐานธรรมชาติ (ฐาน  $e$ )

รูปแบบคำสั่ง: `log(x)`

พารามิเตอร์:  $x$  คือจำนวนเต็ม หรือจำนวนจริง โดย  $x$  ต้องมีค่ามากกว่า 0

ค่าที่ส่งกลับ: ค่าลอการิทึมฐานธรรมชาติ

ตัวอย่าง:

```
import math # This will import math module
print ("math.log(100.12) : ", math.log(100.12))
print ("math.log(100.72) : ", math.log(100.72))
print ("math.log(math.pi) : ", math.log(math.pi))
```



**OUTPUT**

```
math.log(100.12) : 4.6063694665635735
math.log(100.72) : 4.612344389736092
math.log(math.pi) : 1.1447298858494002
```

- **log10()** คำนวณหาค่าลอการิทึมฐานสิบ

รูปแบบคำสั่ง: `log10(x)`

พารามิเตอร์:  $x$  คือจำนวนเต็ม หรือจำนวนจริง โดย  $x$  ต้องมีค่ามากกว่า 0

ค่าที่ส่งกลับ: ค่าลอการิทึมฐานสิบ

ตัวอย่าง:

```
import math # This will import math module
print ("math.log10(100.12):", math.log10(100.12))
print ("math.log10(100.72):", math.log10(100.72))
print ("math.log10(math.pi):", math.log10(math.pi))
```



**OUTPUT**

```
math.log10(100.12) : 2.0005208409361854
math.log10(100.72) : 2.003115717099806
math.log10(math.pi) : 0.49714987269413385
```

- **max()** คำนวณหาเลขจำนวนที่มีค่ามากที่สุดในสมาชิกทั้งหมด

รูปแบบคำสั่ง: `max(x1, x2,...)`

พารามิเตอร์:  $x_1, x_2, \dots, x_n$  คือจำนวนเต็ม หรือจำนวนจริง

ค่าที่ส่งกลับ: เลขจำนวนที่มีค่ามากที่สุด

ตัวอย่าง:

```
print ("max(80, 100, 1000) : ", max(80, 100, 1000))
print ("max(-20, 100, 400) : ", max(-20, 100, 400))
print ("max(-80, -20, -10) : ", max(-80, -20, -10))
print ("max(0, 100, -400) : ", max(0, 100, -400))
```



**OUTPUT**

```
max(80, 100, 1000) : 1000
max(-20, 100, 400) : 400
max(-80, -20, -10) : -10
max(0, 100, -400) : 100
```

- **min()** คำนวณหาเลขจำนวนที่มีค่าน้อยที่สุดในสมาชิกทั้งหมด

รูปแบบคำสั่ง: `min(x1, x2,...)`

พารามิเตอร์:  $x_1, x_2, \dots, x_n$  คือจำนวนเต็ม หรือจำนวนจริง

ค่าที่ส่งกลับ: เลขจำนวนที่มีค่าน้อยที่สุด

ตัวอย่าง:

```
print ("min(80, 100, 1000) : ", min(80, 100, 1000))
print ("min(-20, 100, 400) : ", min(-20, 100, 400))
print ("min(-80, -20, -10) : ", min(-80, -20, -10))
```

```
print ("min(0, 100, -400) : ", min(0, 100, -400))
```



**OUTPUT**

```
min(80, 100, 1000) : 80
min(-20, 100, 400) : -20
min(-80, -20, -10) : -80
min(0, 100, -400) : -400
```

- **modf()** แยกข้อมูลของจำนวนจริงออกเป็นสองส่วนคือ ส่วนหน้าจุดทศนิยม และหลังทศนิยม

ตัวอย่าง:

รูปแบบคำสั่ง: modf(x)

พารามิเตอร์: x คือจำนวนจริงใดๆ

ค่าที่ส่งกลับ: ข้อมูลชนิดทัพเพลล (tuple) มีสมาชิก 2 ค่า โดยสมาชิกตัวแรกเก็บค่าหลังจุดทศนิยม และสมาชิกตัวที่ 2 เก็บจำนวนเต็มหน้าทศนิยม

ตัวอย่าง:

```
import math # This will import math module
print ("math.modf(100.12) : ", math.modf(100.12))
print ("math.modf(100.72) : ", math.modf(100.72))
print ("math.modf(math.pi) : ", math.modf(math.pi))
temp = math.modf(100.12)
print("Integer : ",temp[1])
print("Fractional : ",temp[0])
```



**OUTPUT**

```
math.modf(100.12) : (0.1200000000000455, 100.0)
math.modf(100.72) : (0.719999999999989, 100.0)
math.modf(math.pi) : (0.14159265358979312, 3.0)
Integer: 100.0
Fractional: 0.1200000000000455
```

- **pow()** คำนวณเลขยกกำลัง

รูปแบบคำสั่ง: pow(x, y)

พารามิเตอร์: ฐานคือ x และ กำลังคือ y ( $x^y$ )

ค่าที่ส่งกลับ: เลขจำนวนจริงที่ยกกำลังแล้ว

ตัวอย่าง:

```
import math # This will import math module
print ("math.pow(100, 2) : ", math.pow(100, 2))
print ("math.pow(100, -2) : ", math.pow(100, -2))
print ("math.pow(2, 4) : ", math.pow(2, 4))
print ("math.pow(3, 0) : ", math.pow(3, 0))
```



**OUTPUT**

```
math.pow(100, 2) : 10000.0
math.pow(100, -2) : 0.0001
math.pow(2, 4) : 16.0
math.pow(3, 0) : 1.0
```

- **round()** ปัดเศษตัวเลขตามจำนวนหลักที่ระบุ

รูปแบบคำสั่ง: round(x [,n])

พารามิเตอร์:  $x$  คือตัวเลขที่ต้องการปัด และ  $n$  คือจำนวนทศนิยมที่ต้องแสดงผล เมื่อไม่กำหนดค่า  $n$  จะปัดเป็นจำนวนเต็มแทน  
 ค่าที่ส่งกลับ: เลขจำนวนที่ได้รับการปัดเศษแล้ว  
 ตัวอย่าง:

```
print ("round(20.1234) : ", round(20.1234))
print ("round(80.23456, 2) : ", round(80.23456, 2))
print ("round(100.000056, 3) : ", round(100.000056, 3))
print ("round(-100.000056, 3) : ", round(-100.000056, 3))
```



**OUTPUT**

```
round(20.1234) : 20
round(80.23456, 2) : 80.23
round(100.000056, 3) : 100.0
round(-100.000056, 3) : -100.0
```

- **sqrt()** คำนวณหารากที่สอง (Square root)

รูปแบบคำสั่ง:  $\text{sqrt}(x)$

พารามิเตอร์:  $x$  คือเลขจำนวนเต็ม หรือจำนวนจริง

ค่าที่ส่งกลับ: เลขจำนวนที่ได้รับการคำนวณหารากที่สองแล้ว

ตัวอย่าง:

```
import math # This will import math module
print ("math.sqrt(100) : ", math.sqrt(100))
print ("math.sqrt(7) : ", math.sqrt(7))
print ("math.sqrt(math.pi) : ", math.sqrt(math.pi))
```



**OUTPUT**

```
math.sqrt(100) : 10.0
math.sqrt(7) : 2.6457513110645907
math.sqrt(math.pi) : 1.7724538509055159
```

## ฟังชันตรีโกณมิติ (Random Number Functions)

- **choice()** เลือกข้อมูลที่ได้จากการสุ่มค่า

รูปแบบคำสั่ง:  $\text{choice(seq)}$

พารามิเตอร์:  $\text{seq}$  เป็นข้อมูลที่ต้องการใช้สุ่ม เช่น ลิสต์ หรือทัพเพิล

ค่าที่ส่งกลับ: ข้อมูลที่ได้รับการสุ่มเพียง 1 ค่าเท่านั้น

ตัวอย่าง:

```
import random # import random class
print("choice([1,2,3,5,9]) : ", random.choice([1,2,3,5,9]))
print("choice('A String') : ", random.choice('A String'))
```



**OUTPUT**

```
choice([1, 2, 3, 5, 9]) : 1
choice('A String') : i
```

- **randrange()** สุ่มค่าข้อมูลเป็นช่วง

รูปแบบคำสั่ง: `randrange ([start,] stop [,step])`

พารามิเตอร์: `start` คือการกำหนดข้อมูลเริ่มต้น, `stop` คือสิ้นสุดข้อมูล และ `step` คือความห่างของข้อมูล เช่น กำหนดค่าเริ่มต้นเท่ากับ 1 ค่าสิ้นสุดเท่ากับ 10 และกำหนดความห่างเป็น 2 จะได้ผลลัพธ์ที่เกิดจากการสร้างชุดของข้อมูลคือ 1, 3, 4, 7, 9 แต่ชุดข้อมูลดังกล่าวจะถูกสุ่มเลือกออกมาเพียง 1 ค่าเท่านั้น

ค่าที่ส่งกลับ: ข้อมูลที่ได้รับการสุ่มเพียง 1 ค่าเท่านั้น

ตัวอย่าง:

```
import random
Select an even number in 100 <= number < 1000
print("randrange(100,1000,2):",random.randrange(100,1000,2))
Select another number in 100 <= number < 1000
print("randrange(100,1000,3):",random.randrange(100,1000,3))
```



`randrange(100, 1000, 2) : 612  
randrange(100, 1000, 3) : 334`

OUTPUT

- **random()** สุ่มค่าข้อมูลเลขจำนวนจริง ที่มีค่าระหว่าง 0 – 1

รูปแบบคำสั่ง: `random()`

พารามิเตอร์: ไม่มี

ค่าที่ส่งกลับ: จำนวนจริงที่ได้รับการสุ่มเพียง 1 ค่าเท่านั้น

ตัวอย่าง:

```
import random
First random number
print ("random() : ", random.random())
Second random number
print ("random() : ", random.random())
```



`random() : 0.6819560436956998  
random() : 0.13485861760609497`

OUTPUT

- **seed()** กำหนดค่าเริ่มต้นก่อนเรียกใช้ฟังชัน `random()` เพื่อให้ค่าสุ่มที่ได้มีค่าเท่าเดิม

รูปแบบคำสั่ง: `seed([x])`

พารามิเตอร์: `x` เป็นจำนวนเต็ม หรือจริง เมื่อต้องการได้ค่าสุ่มที่เท่าเดิมทุกๆ ครั้ง ให้กำหนดเป็นตัวเลขที่เหมือนกัน

ค่าที่ส่งกลับ: จำนวนจริงที่ได้รับการสุ่มเพียง 1 ค่าเท่านั้น มีค่าระหว่าง 0 - 1

ตัวอย่าง:

```
import random
random.seed(5)
print("Random number with seed 5:",random.random())
```

```
It will generate same random number
random.seed(10)
print("Random number with seed 10:", random.random())
random.seed(10)
print("Random number with seed 10:", random.random())
```



**OUTPUT**

```
Random number with seed 5 : 0.6229016948897019
Random number with seed 10 : 0.5714025946899135
Random number with seed 10 : 0.5714025946899135
```

- **shuffle()** สลับตำแหน่งของข้อมูลในรายการของตัวแปรชนิดลิสต์ (list)

รูปแบบคำสั่ง: `shuffle(lst)`

พารามิเตอร์: `lst` คือข้อมูลชนิดลิสต์ที่ต้องการสลับตำแหน่ง

ค่าที่ส่งกลับ: ข้อมูลชนิดลิสต์ที่ได้รับการสลับตำแหน่งแล้ว

ตัวอย่าง:

```
import random
list = [20, 16, 10, 5];
print ("Original list : ", list)
random.shuffle(list)
print ("Reshuffled list : ", list)
random.shuffle(list)
print ("Reshuffled list : ", list)
```



**OUTPUT**

```
Original list : [20, 16, 10, 5]
Reshuffled list : [5, 16, 10, 20]
Reshuffled list : [16, 20, 5, 10]
```

- **uniform()** สุ่มค่าเลขจำนวนจริง

รูปแบบคำสั่ง: `uniform(x, y)`

พารามิเตอร์: `x` คือขอบเขตเริ่มต้นข้อมูล และ `y` คือขอบเขตสิ้นสุดของข้อมูล ค่าที่สุ่มได้จะอยู่ระหว่างช่วง `x` ถึง `y` เพียง 1 ค่าเท่านั้น

ค่าที่ส่งกลับ: จำนวนจริงที่ได้รับการสุ่มเพียง 1 ค่าเท่านั้น

ตัวอย่าง:

```
import random
print("Random Float uniform(5 - 10):", random.uniform(5, 10))
print("Random Float uniform(7 - 14):", random.uniform(7, 14))
```



**OUTPUT**

```
Random Float uniform(5 - 10) : 6.6396969456700115
Random Float uniform(7 - 14) : 10.61079139843364
```

## พังชันตรีโกณมิติ (Trigonometric Functions)

- **acos()** คำนวณค่า arc cosine ของ `x`

รูปแบบคำสั่ง: `acos(x)`

พารามิเตอร์:  $x$  คือเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า arc cosine ที่ได้เป็น radians อยู่ในช่วง -1 ถึง 1

ตัวอย่าง:

```
import math
print ("acos(0.64) : ", math.acos(0.64))
print ("acos(0) : ", math.acos(0))
print ("acos(-1) : ", math.acos(-1))
print ("acos(1) : ", math.acos(1))
```



**OUTPUT**

|              |                    |
|--------------|--------------------|
| acos(0.64) : | 0.8762980611683406 |
| acos(0) :    | 1.5707963267948966 |
| acos(-1) :   | 3.141592653589793  |
| acos(1) :    | 0.0                |

- **asin()** คำนวณค่า arc sine ของ  $x$

รูปแบบคำสั่ง: `asin(x)`

พารามิเตอร์:  $x$  คือเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า arc sine ที่ได้เป็น radians อยู่ในช่วง -1 ถึง 1

ตัวอย่าง:

```
import math
print ("asin(0.64) : ", math.asin(0.64))
print ("asin(0) : ", math.asin(0))
print ("asin(-1) : ", math.asin(-1))
print ("asin(1) : ", math.asin(1))
```



**OUTPUT**

|              |                     |
|--------------|---------------------|
| asin(0.64) : | 0.694498265626556   |
| asin(0) :    | 0.0                 |
| asin(-1) :   | -1.5707963267948966 |
| asin(1) :    | 1.5707963267948966  |

- **atan()** คำนวณค่า arc tangent ของ  $x$

รูปแบบคำสั่ง: `atan(x)`

พารามิเตอร์:  $x$  คือเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า arc tangent ที่ได้เป็น radians

ตัวอย่าง:

```
import math
print ("atan(0.64) : ", math.atan(0.64))
print ("atan(0) : ", math.atan(0))
print ("atan(10) : ", math.atan(10))
print ("atan(-1) : ", math.atan(-1))
print ("atan(1) : ", math.atan(1))
```



**OUTPUT**

|              |                    |
|--------------|--------------------|
| atan(0.64) : | 0.5693131911006619 |
| atan(0) :    | 0.0                |
| atan(10) :   | 1.4711276743037347 |

```
atan(-1) : -0.7853981633974483
atan(1) : 0.7853981633974483
```

- **atan2()** คำนวณค่า  $\text{atan}(y / x)$  ค่าที่ได้เป็น radians

รูปแบบคำสั่ง: `atan2(y, x)`

พารามิเตอร์:  $x$  และ  $y$  เป็นเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า  $\text{atan2}(y / x)$  ที่ได้เป็น radian

ตัวอย่าง:

```
import math
print ("atan2(-0.50,-0.50) : ", math.atan2(-0.50,-0.50))
print ("atan2(0.50,0.50) : ", math.atan2(0.50,0.50))
print ("atan2(5,5) : ", math.atan2(5,5))
print ("atan2(-10,10) : ", math.atan2(-10,10))
print ("atan2(10,20) : ", math.atan2(10,20))
```



```
atan2(-0.50,-0.50) : -2.356194490192345
atan2(0.50,0.50) : 0.7853981633974483
atan2(5,5) : 0.7853981633974483
atan2(-10,10) : -0.7853981633974483
atan2(10,20) : 0.4636476090008061
```

**OUTPUT**

- **cos()** คำนวณค่า cosine ของ  $x$

รูปแบบคำสั่ง: `acos(x)`

พารามิเตอร์:  $x$  คือเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า cosine ที่ได้เป็น radians อยู่ในช่วง  $-1 \leq x \leq 1$

ตัวอย่าง:

```
import math
print ("cos(3) : ", math.cos(3))
print ("cos(-3) : ", math.cos(-3))
print ("cos(0) : ", math.cos(0))
print ("cos(math.pi) : ", math.cos(math.pi))
print ("cos(2*math.pi) : ", math.cos(2*math.pi))
```



```
cos(3) : -0.9899924966004454
cos(-3) : -0.9899924966004454
cos(0) : 1.0
cos(math.pi) : -1.0
cos(2*math.pi) : 1.0
```

**OUTPUT**

- **hypot()** คำนวณค่า Euclidean norm

รูปแบบคำสั่ง: `hypot(x, y)`

พารามิเตอร์: โดยค่า  $x$  และ  $y$  เป็นเลขจำนวนเต็ม

ค่าที่ส่งกลับ: ค่า Euclidean norm ที่ถูกคำนวณด้วยสมการ  $\sqrt{x^2 + y^2}$

ตัวอย่าง:

```
import math
print ("hypot(3, 2) : ", math.hypot(3, 2))
```

```
print ("hypot(-3, 3) : ", math.hypot(-3, 3))
print ("hypot(0, 2) : ", math.hypot(0, 2))
```



**hypot(3, 2) : 3.605551275463989**

**hypot(-3, 3) : 4.242640687119285**

**OUTPUT hypot(0, 2) : 2.0**

- **sin()** คำนวณค่า sine ของ x

รูปแบบคำสั่ง: sin(x)

พารามิเตอร์: x คือเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า sine ที่ได้เป็น radians อยู่ในช่วง -1 ถึง 1

ตัวอย่าง:

```
import math
print ("sin(3) : ", math.sin(3))
print ("sin(-3) : ", math.sin(-3))
print ("sin(0) : ", math.sin(0))
print ("sin(math.pi) : ", math.sin(math.pi))
print ("sin(math.pi/2) : ", math.sin(math.pi/2))
```



**sin(3) : 0.1411200080598672**

**sin(-3) : -0.1411200080598672**

**OUTPUT sin(0) : 0.0**

**sin(math.pi) : 1.2246467991473532e-16**

**sin(math.pi/2) : 1.0cos(2\*math.pi) : 1.0**

- **tan()** คำนวณค่า tangent ของ x

รูปแบบคำสั่ง: tan(x)

พารามิเตอร์: x คือเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า arc cosine ค่าที่ได้เป็น radians อยู่ในช่วง -1 ถึง 1

ตัวอย่าง:

```
import math
print ("tan(3) : ", math.tan(3))
print ("tan(-3) : ", math.tan(-3))
print ("tan(0) : ", math.tan(0))
print ("tan(math.pi) : ", math.tan(math.pi))
print ("tan(math.pi/2) : ", math.tan(math.pi/2))
print ("tan(math.pi/4) : ", math.tan(math.pi/4))
```



**tan(3) : -0.1425465430742778**

**tan(-3) : 0.1425465430742778**

**OUTPUT tan(0) : 0.0**

**tan(math.pi) : -1.2246467991473532e-16**

**tan(math.pi/2) : 1.633123935319537e+16**

**tan(math.pi/4) : 0.9999999999999999**

- **degrees()** แปลงค่าจาก radians เป็น degree

รูปแบบคำสั่ง: degrees(x)

พารามิเตอร์: x เป็นเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า degree ของ radians x

ตัวอย่าง:

```
import math
print ("degrees(3) : ", math.degrees(3))
print ("degrees(-3) : ", math.degrees(-3))
print ("degrees(0) : ", math.degrees(0))
print ("degrees(math.pi) : ", math.degrees(math.pi))
print ("degrees(math.pi/2) : ", math.degrees(math.pi/2))
print ("degrees(math.pi/4) : ", math.degrees(math.pi/4))
```



**OUTPUT**

```
degrees(3) : 171.88733853924697
degrees(-3) : -171.88733853924697
degrees(0) : 0.0
degrees(math.pi) : 180.0
degrees(math.pi/2) : 90.0
degrees(math.pi/4) : 45.0
```

- **radians()** แปลงค่าจาก degree เป็น radians

รูปแบบคำสั่ง: radians(x)

พารามิเตอร์: x เป็นเลขจำนวนใดๆ

ค่าที่ส่งกลับ: ค่า radians ของ degree x

ตัวอย่าง:

```
import math
print ("radians(3) : ", math.radians(3))
print ("radians(-3) : ", math.radians(-3))
print ("radians(0) : ", math.radians(0))
print ("radians(math.pi) : ", math.radians(math.pi))
print ("radians(math.pi/2) : ", math.radians(math.pi/2))
print ("radians(math.pi/4) : ", math.radians(math.pi/4))
```



**OUTPUT**

```
radians(3) : 0.05235987755982989
radians(-3) : -0.05235987755982989
radians(0) : 0.0
radians(math.pi) :
0.05483113556160755
radians(math.pi/2) :
0.027415567780803774
radians(math.pi/4) :
0.013707783890401887
```

## ค่าคงที่ทางคณิตศาสตร์ (Mathematical constants)

ค่าคงที่ที่ใช้ในทางคณิตศาสตร์ ที่นิยมมี 2 ตัวคือ ค่า pi = 3.14159 และ e = 2.71828

math.pi, math.e

## เมธอดและฟังชันที่ดำเนินการกับสตริง (Built-in String Methods)

สมมุติให้ str เป็นสตริงใดๆ เช่น str = "Python" หรือ str = 'Programming'

- **str.capitalize()** แปลงตัวอักษรตัวแรกในสตริงให้เป็นตัวใหญ่

รูปแบบคำสั่ง: str.capitalize()

พารามิเตอร์: ไม่มี

ค่าที่ส่งกลับ: ข้อความสตริงที่มีอักษรตัวแรกเป็นตัวใหญ่

ตัวอย่าง:

```
str = "this is string example....wow!!!"
print ("str.capitalize() : ", str.capitalize())
```



**OUTPUT**

str.capitalize() : This is string example....wow!!!

- **str.center()** จัดเรียงสตริงให้อยู่ตรงกลาง โดยเติมอักษร (fillchar) ด้านซ้าย และด้านขวาของสตริงให้ครบจำนวนความกว้างที่กำหนด (width)

รูปแบบคำสั่ง: str.center(width, fillchar)

พารามิเตอร์: width ความกว้างของสตริง, fillchar คือตัวอักษรที่ต้องการเติมในสตริง

ค่าที่ส่งกลับ: ข้อความสตริงที่อยู่ตรงกลางระหว่างตัวอักษรที่เติมเข้าไป

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print ("str.center(40, 'x'):", str.center(40, 'x'))
```



**OUTPUT**

str.center(40, 'x'):xxxxthis is string example....wow!!!xxxx

- **str.count()** ค้นหาครั้งที่ต้องการ

รูปแบบคำสั่ง: str.count(str, beg= 0,end=len(string))

พารามิเตอร์: str คือกลุ่มคำที่ต้องการค้นหา, beg คือตำแหน่งเริ่มต้นที่ต้องการค้นหา, end คือตำแหน่งสุดท้ายที่ต้องการค้นหา ถ้าไม่ระบุตำแหน่งสุดท้ายของสตริงสามารถใช้ len() เพื่อหาความยาวสตริง (อักษรตัวแรกของสตริงเริ่มต้นที่ 0)

ค่าที่ส่งกลับ: จำนวนนับของกลุ่มคำนั้นๆ

ตัวอย่าง:

```
str = "this is string example....wow!!!";
sub = "i";
print("str.count(sub,4,40):",str.count(sub, 4, 40))
sub = "wow";
print ("str.count(sub):",str.count(sub)))
```



```
str.count(sub, 4, 40) : 2
str.count(sub) : 1
```

OUTPUT

- **str.decode()** ถอดรหัสสตริงที่ถูกเข้ารหัสไว้

รูปแบบคำสั่ง: str.decode(encoding='UTF-8',errors='strict')

พารามิเตอร์: encoding คือแบบของภาษาที่ใช้ถอดรหัส เช่น UTF-8, ISO, Latin เป็นต้น, error คือข้อกำหนดเมื่อเกิดข้อผิดพลาดขึ้นขณะถอดรหัส เช่น strict, ignore, replace สำหรับ strict เป็นค่า default หมายถึงให้สนใจข้อผิดพลาดในการถอดรหัส

ค่าที่ส่งกลับ: ข้อความที่ถูกถอดรหัสแล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
This is encoding
str = str.encode('base64','strict');
print("Encoded String:" + str)
print("Decoded String:" + str.decode('base64','strict'))
```



OUTPUT

```
Encoded String: dGhpcyBpcyBzdHJpbmcgZXhhbXBsZS4uLi53b3chISE=
Decoded String: this is string example....wow!!!
```

- **str.encode()** เข้ารหัสสตริง

รูปแบบคำสั่ง: str.encode(encoding='UTF-8',errors='strict')

พารามิเตอร์: encoding คือแบบอักษรที่ใช้เข้ารหัส เช่น UTF-8, ISO, Latin เป็นต้น, error คือข้อกำหนดเมื่อเกิดข้อผิดพลาดขึ้นขณะเข้ารหัส เช่น strict, ignore, replace สำหรับ strict เป็นค่า default หมายถึงให้สนใจข้อผิดพลาดในการเข้ารหัส

ค่าที่ส่งกลับ: ข้อความที่ถูกเข้ารหัสแล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print("Encoded String:" + str.encode('base64','strict'))
print("Decoded String:" + str.decode('base64','strict'))
```



OUTPUT

```
Encoded String: dGhpcyBpcyBzdHJpbmcgZXhhbXBsZS4uLi53b3chISE=
```

- **str.endswith()** ค้นหากรุ่มคำ โดยเริ่มต้นจากทางด้านท้ายของสตริง

รูปแบบคำสั่ง: str.endswith(suffix[, start[, end=len(string)]])

พารามิเตอร์: suffix คือกลุ่มคำที่ต้องการค้นหา, start คือตำแหน่งเริ่มต้น, end คือขนาดความยาวของสตริง

ค่าที่ส่งกลับ: ถ้าค้นหาพบจะส่งค่ากลับเป็นจริง (True)

ตัวอย่าง:

```
str = "this is string example....wow!!!";
suffix = "wow!!!";
print (str.endswith(suffix));
print (str.endswith(suffix,20));
suffix = "is";
print (str.endswith(suffix, 2, 4));
print (str.endswith(suffix, 2, 6));
```

|                                                                                                    |                               |
|----------------------------------------------------------------------------------------------------|-------------------------------|
| <br><b>OUTPUT</b> | True<br>True<br>True<br>False |
|----------------------------------------------------------------------------------------------------|-------------------------------|



**Note:** ควรจำไว้ว่า สัญลักษณ์ [ , start[], end]] หมายถึง ค่า start และ end จะใส่หรือไม่ใส่ก็ได้

- **str.expandtabs()** แทนที่แท็บ (\t) ในสตริงด้วยช่องว่าง (space) โดยค่า default ของช่องว่างที่แทนที่คือ 1 tab ต่อ 1 space (เมื่อไม่กำหนด tabsize) แต่ถ้าต้องการให้ช่องว่างกว้างขึ้นให้กำหนด tabsize ในอัตราส่วน 1 tab ต่อ 8 spaces โดยค่าตัวเลขที่ป้อนให้คือ 8, 16, 24, 32 เป็นต้น

รูปแบบคำสั่ง: str.expandtabs(tabsize)

พารามิเตอร์: ความกว้างของ tabsize

ค่าที่ส่งกลับ: ข้อความที่แทนที่ ด้วยช่องว่างตามขนาดของ tabsize

ตัวอย่าง:

```
str = "this is\tstring example....wow!!!";
print("Original string:" + str)
print("Defualt exapanded tab:" + str.expandtabs())
print("Double exapanded tab:" + str.expandtabs(16))
```



**OUTPUT**

```
Original string: this is string example....wow!!!
Defualt exapanded tab: this is string example....wow!!!
Double exapanded tab: this is string example....wow!!!
```

- **str.find()** ค้นหากลุ่มคำในสตริง

รูปแบบคำสั่ง: str.find(str, beg=0 end=len(string))

พารามิเตอร์: str คือกลุ่มคำที่ต้องการค้นหา, beg คือตำแหน่งเริ่มต้น (default = 0), end คือขนาดความยาวของสตริง (ถ้าไม่กำหนดถือว่าค้นหาหมด)

ค่าที่ส่งกลับ: ถ้าค้นหาพบคำพจน์คำที่ต้องการจะส่งตำแหน่งเริ่มต้นของกลุ่มคำกลับคืนมา ถ้าไม่พบผลลัพธ์คือ -1

ตัวอย่าง:

```
str1 = "this is string example....wow!!!";
str2 = "exam";
print (str1.find(str2));
print (str1.find(str2, 10));
print (str1.find(str2, 40));
print (str1.find(str2,0,len(str1)));)
```



15  
15  
-1  
15

**OUTPUT**

- **str.rfind()** การทำงานเหมือน **find()** แต่ผลลัพธ์ที่ส่งกลับมาคือ ตำแหน่งของกลุ่มคำที่ต้องการค้นหาลำดับสุดท้าย

รูปแบบคำสั่ง: `str.rfind(str, beg=0,end=len(string))`

พารามิเตอร์: str คือกลุ่มคำที่ต้องการค้นหา, beg คือตำแหน่งเริ่มต้น (default = 0), end คือขนาดความยาวของสตริง (ถ้าไม่กำหนดถือว่าค้นหาหมด)

ค่าที่ส่งกลับ: ถ้าค้นหาพบคำพจน์คำที่ต้องการจะส่งตำแหน่งเริ่มต้นของกลุ่มคำในลำดับสุดท้ายกลับคืนมา ถ้าไม่พบผลลัพธ์คือ -1

ตัวอย่าง:

```
str1 = "this is string example....wow!!!";
str2 = "exam";
print (str1.find(str2));
print (str1.find(str2, 10));
print (str1.find(str2, 40));
print (str1.find(str2,0,len(str1)));)
```



24  
5  
24  
-1

**OUTPUT**

- **str.index()** ทำงานเหมือน **find()** ทุกประการ ต่างกันคือ ถ้าค้นหาพบกลุ่มคำไม่เจอจะส่งค่ากลับเป็น exception ออกมาแทน

รูปแบบคำสั่ง: `str.index(str, beg=0, end=len(string))`

พารามิเตอร์: str คือกลุ่มคำที่ต้องการค้นหา, beg คือตำแหน่งเริ่มต้น (default = 0), end คือขนาดความยาวของสตริง (ถ้าไม่กำหนดถือว่าค้นหาหมด)

ค่าที่ส่งกลับ: ถ้าค้นหาพบคำพจน์คำที่ต้องการจะส่งตำแหน่งเริ่มต้นของกลุ่มคำกลับคืนมา ถ้าค้นหาพบกลุ่มคำไม่เจอจะส่งค่ากลับเป็น exception

ตัวอย่าง:

```
str1 = "this is string example....wow!!!";
str2 = "exam";
print (str1.index(str2));
print (str1.index(str2, 10));
print (str1.index(str2, 40));
```



**OUTPUT**

```
15
15
Traceback (most recent call last):
 File "C:/Python33/testx.py", line 5, in <module>
 print (str1.index(str2, 40));
ValueError: substring not found
```

- **str.rindex()** การทำงานเหมือน index() แต่เริ่มทำการค้นหาจากด้านท้ายสตริงแทน

รูปแบบคำสั่ง: str.rindex(str, beg=0, end=len(string))

พารามิเตอร์: str คือกลุ่มคำที่ต้องการค้นหา, beg คือตำแหน่งเริ่มต้น (default = 0), end คือขนาดความยาวของสตริง (ถ้าไม่กำหนดถือว่าค้นหาหมด)

ค่าที่ส่งกลับ: ถ้าค้นหาพบจะส่งตำแหน่งเริ่มต้นของกลุ่มคำที่ค้นหาจากด้านท้ายของสตริงกลับคืนมา ถ้าค้นหาไม่เจอจะส่งค่ากลับเป็น exception

ตัวอย่าง:

```
str1 = "this is string example....wow!!!";
str2 = "is";
print(str1.rindex(str2)); # This method of rindex()
print(str1.index(str2)); # This method of index()
```



**OUTPUT**

5

2

- **str.isalnum(), str.isalpha()** ตรวจสอบว่าสตริงเป็น Alphanumeric (สัญลักษณ์ต่างๆ ที่พิมพ์จากแป้นพิมพ์ ทั้งตัวอักษร และตัวเลข โดยไม่รวมช่องว่าง และสัญลักษณ์พิเศษ) หรือไม่

รูปแบบคำสั่ง: str.isalnum(), str.isalpha()

พารามิเตอร์: str คือกลุ่มคำที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: ถ้าเจออักษรที่ไม่ใช่ Alphanumeric เพียงตัวเดียว จะส่งค่ากลับเป็นเท็จ

ตัวอย่าง:

```
str = "this2009"; # No space in this string
print (str.isalnum());
str = "this is string example....wow!!!";
print (str.isalnum());
```

 **OUTPUT**

|       |
|-------|
| True  |
| False |

- **str.isdigit()** ตรวจสอบตัวเลขในสตริง

รูปแบบคำสั่ง: `str.isdigit()`

พารามิเตอร์: `str` คือสตริงที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อข้อมูลในสตริงทุกตัวเป็นตัวเลขทั้งหมด

ตัวอย่าง:

```
str = "123456"; # Only digit in this string
print (str.isdigit());
str = "this is string example....555 !!!";
print (str.isdigit());
```

 **OUTPUT**

|       |
|-------|
| True  |
| False |

- **str.islower()** ตรวจสอบว่าตัวอักษรในสตริงเป็นตัวอักษรตัวเล็กหรือไม่

รูปแบบคำสั่ง: `str.islower()`

พารามิเตอร์: `str` คือสตริงที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อข้อมูลในสตริงทุกตัวเป็นอักษรตัวเล็กทั้งหมด

ตัวอย่าง:

```
str = "THIS is string example....wow!!!";
print (str.islower());
str = "this is string example....wow!!!";
print (str.islower());
```

 **OUTPUT**

|       |
|-------|
| False |
| True  |

- **str.isnumeric()** ตรวจสอบตัวเลขในสตริงชนิด Unicode

รูปแบบคำสั่ง: `str.isnumeric()`

พารามิเตอร์: `str` คือสตริงที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อข้อมูลในสตริงชนิด Unicode ทุกตัวเป็นตัวเลขทั้งหมด

ตัวอย่าง:

```
str = u"this2009"; # u means Unicode
print (str.isnumeric());
str = u"23443434";
print (str.isnumeric());
```

 **OUTPUT**

|       |
|-------|
| False |
| True  |

- **str.isspace()** ตรวจสอบสตริงว่า

รูปแบบคำสั่ง: str.isspace()

พารามิเตอร์: str คือสตริงที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อข้อมูลในสตริงเป็นค่าว่าง

ตัวอย่าง:

```
str = " ";
print (str.isspace());
str = "This is string example....wow!!!";
print (str.isspace());
```

|                                                                                   |       |
|-----------------------------------------------------------------------------------|-------|
|  | True  |
|                                                                                   | False |

OUTPUT

- **str.istitle()** ตรวจสอบอักษรตัวแรกของข้อความว่าเป็นตัวใหญ่หรือไม่

รูปแบบคำสั่ง: str.istitle()

พารามิเตอร์: str คือสตริงที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อคำขึ้นต้นในทุก ๆ กลุ่มคำเป็นตัวใหญ่ทั้งหมด

ตัวอย่าง:

```
str = "This Is String Example...Wow!!!";
print (str.istitle());
str = "This is string example....wow!!!";
print (str.istitle());
```

|                                                                                     |       |
|-------------------------------------------------------------------------------------|-------|
|  | True  |
|                                                                                     | False |

OUTPUT

- **str.isupper()** ตรวจสอบว่าตัวอักษรในสตริงเป็นตัวใหญ่ทั้งหมด

รูปแบบคำสั่ง: str.isupper()

พารามิเตอร์: str คือสตริงที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อข้อความในสตริงเป็นอักษรตัวใหญ่ทั้งหมด

ตัวอย่าง:

```
str = "THIS IS STRING EXAMPLE....WOW!!!";
print (str.isupper());
str = "THIS is string example....wow!!!";
print (str.isupper());
```

|                                                                                     |       |
|-------------------------------------------------------------------------------------|-------|
|  | True  |
|                                                                                     | False |

OUTPUT

- **str.lstrip()** ลบช่องว่างหน้าสตริงทั้งหมดออก ถ้าต้องการลบตัวอักษรอื่น ๆ ที่ไม่ใช่ช่องว่าง หน้าสตริงออก ให้กำหนดตัวอักษรที่ต้องการลบกับตัวแปร chars

รูปแบบคำสั่ง: str.lstrip([chars])

พารามิเตอร์: str คือสตริงที่ต้องการลบตัวอักษร, chars คือตัวอักษรที่ต้องการลบทิ้ง

ค่าที่ส่งกลับ: สตริงที่ลบช่องว่าง หรืออักษรที่ต้องการออกแล้ว

ตัวอย่าง:

```
str = " this is string example....wow!!! ";
print (str.lstrip());
str = "55555this is string example....wow!!!55555";
print (str.lstrip('5'));
```



```
this is string example....wow!!!
this is string example....wow!!!55555
```

- **str.rstrip()** ทำงานเหมือน lstrip() แตกต่างกันตรงที่ rstrip() จะแทนที่ตัวอักษรมาจากด้านท้ายของสตริงแทน

รูปแบบคำสั่ง: str.rstrip([chars])

พารามิเตอร์: str คือสตริงที่ต้องการลบตัวอักษร, chars คือตัวอักษรที่ต้องการลบทิ้ง

ค่าที่ส่งกลับ: สตริงที่ลบช่องว่าง หรืออักษรที่ต้องการออกแล้ว

ตัวอย่าง:

```
str = " this is string example....wow!!! ";
print (str.rstrip());
str = "55555this is string example....wow!!!55555";
print (str.rstrip('5'));
```



```
this is string example....wow!!!
55555this is string example....wow!!!
```

- **str.join()** เชื่อมกลุ่มของสตริงเข้าด้วยกัน โดยคั่นด้วยเครื่องหมายที่ผู้ใช้กำหนดเอง

รูปแบบคำสั่ง: str.join(seq)

พารามิเตอร์: str คือสตริงชุดที่ 1 ใช้สำหรับคั่นระหว่างสตริง, seq สตริงชุดที่ 2

ค่าที่ส่งกลับ: สตริงที่เกิดจากการเชื่อมต่อระหว่าง str และ seq

ตัวอย่าง:

```
str = "/";
This is concatenation fo string
seq = ("12", "12", "2012"); # Sequence of strings.
print (str.join(seq));
```



```
12/12/2012
```

- **len()** คำนวณหาความยาวของสตริง

รูปแบบคำสั่ง: len(string)

พารามิเตอร์: สตริงที่ต้องการหาความยาว

ค่าที่ส่งกลับ: ความยาวของสตริง

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print ("Length of the string: ", len(str));
```



Length of the string: 32

- **str.ljust()** เติมตัวอักษรลงในด้านท้ายสตริงให้ครบตามจำนวนที่กำหนดไว้

รูปแบบคำสั่ง: str.ljust(width[, fillchar])

พารามิเตอร์: width คือความยาวของสตริงที่ต้องการ, fillchar คือตัวอักษรที่ต้องการเติมให้เต็ม ถ้าไม่กำหนด fillcahr ไว้ จะมีค่าเป็นช่องว่างโดยอัตโนมัติ

ค่าที่ส่งกลับ: สตริงที่เติมตัวอักษรตามที่ต้องการแล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print (str.ljust(50, '@'));
```



this is string example....wow!!!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

- **str.rjust()** ทำงานเหมือน ljust() แต่ทำการเติมอักษรจากด้านหน้าของสตริงแทน

รูปแบบคำสั่ง: str.rjust(width[, fillchar])

พารามิเตอร์: width คือความยาวของสตริงที่ต้องการ, fillchar คือตัวอักษรที่ต้องการเติมให้เต็ม ถ้าไม่กำหนด fillcahr ไว้ จะมีค่าเป็นช่องว่างโดยอัตโนมัติ

ค่าที่ส่งกลับ: สตริงที่เติมตัวอักษรตามที่ต้องการแล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print (str.rjust(50, '@'));
```



@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@this is string example....wow!!!

- **str.lower()** เปลี่ยนตัวอักษรในสตริงให้เป็นอักษรตัวเล็กทั้งหมด

รูปแบบคำสั่ง: str.lower()

พารามิเตอร์: str คือสตริงที่ต้องการเปลี่ยนให้เป็นตัวเล็ก

ค่าที่ส่งกลับ: สตริงที่เป็นตัวอักษรตัวเล็กทั้งหมดแล้ว

ตัวอย่าง:

```
str = "THIS IS STRING EXAMPLE....WOW!!!";
print (str.lower());
```



**OUTPUT**

this is string example....wow!!!

- **max()** ค้นหาตัวอักษรที่มีค่าสูงที่สุดจากการเรียงลำดับตัวอักษรของแต่ละภาษา

รูปแบบคำสั่ง: max(str)

พารามิเตอร์: str คือสตริงที่ต้องการค้นหา

ค่าที่ส่งกลับ: ตัวอักษรที่มีค่าสูงที่สุดของแต่ละภาษา

ตัวอย่าง:

```
str = "this is really a string example....wow!!!";
print ("Max character: " + max(str));
str = "this is a string example....wow!!!";
print ("Max character: " + max(str));
str = "กขคงจพศօອ"
print ("Max character of Thai is: " + max(str));
```



**OUTPUT**

Max character: y

Max character: x

Max character of Thai is: อ

- **min()** ค้นหาตัวอักษรที่มีค่าต่ำที่สุดจากการเรียงลำดับตัวอักษรของแต่ละภาษา

รูปแบบคำสั่ง: min(str)

พารามิเตอร์: str คือสตริงที่ต้องการค้นหา

ค่าที่ส่งกลับ: ตัวอักษรที่มีค่าต่ำที่สุดของแต่ละภาษา

ตัวอย่าง:

```
str = "this-is-real-string-example....wow!!!";
print ("Min character: " + min(str));
str = "this-is-a-string-example....wow";
print ("Min character: " + min(str));
str = "กขคงจพศօອ"
print ("Max character of Thai is: " + min(str))
```



**OUTPUT**

Min character: !

Min character: -

Max character of Thai is: ก

- **str.replace()** แทนที่ตัวอักษร

รูปแบบคำสั่ง: str.replace(old, new[, max])

พารามิเตอร์: str คือสตริงที่ต้องการแทนที่, old ตัวอักษรเก่าที่ถูกแทนที่, new

ตัวอักษรใหม่ที่ต้องการแทนที่ ถ้าไม่กำหนดค่า max จะทำการแทนที่ทั้งสตริง

ค่าที่ส่งกลับ: стрิงที่ถูกแทนที่แล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!! this is really string";
print (str.replace("is", "was"));
print (str.replace("is", "was", 3));
```



**OUTPUT**

```
thwas was string example....wow!!! thwas was really string
thwas was string example....wow!!! thwas is really string
```

- **str.split()** แยกกลุ่มคำออกจากสตริง โดย default จะทำการแยกโดยพิจารณาจากอักษรว่าง (space) ที่คั่นละห่วงกกลุ่มคำ (และตัดสัญลักษณ์พิเศษออกด้วย เช่น \n) และสามารถแยกกลุ่มคำด้วยตัวอักษรอื่นๆ ที่ผู้ใช้กำหนดได้

รูปแบบคำสั่ง: str.split(strs="", num=string.count(str))

พารามิเตอร์: str คือสตริงที่ต้องการแยกออก, strs ตัวอักษรที่ใช้แยกกลุ่มคำ, num ใช้สำหรับกำหนดจำนวนกลุ่มคำย่อยว่าต้องการทิ้งหมดกี่ชุด ถ้าไม่ใส่จะแยกทุกกลุ่มคำในสตริง

ค่าที่ส่งกลับ: ข้อมูลชนิดลิสต์ของสตริงที่ถูกแยกแล้ว

ตัวอย่าง:

```
str = "Line1-abcdef \nLine2-abc \nLine4-abcd";
str1 = "This + is + program + for + splitting + word";
print (str.split());
print (str.split(' ', 1));
print (str.split('\n'));
print (str1.split(' + '));
```



**OUTPUT**

```
['Line1-abcdef', 'Line2-abc', 'Line4-abcd']
['Line1-abcdef', '\nLine2-abc \nLine4-abcd']
['Line1-abcdef ', 'Line2-abc ', 'Line4-abcd ']
['This', 'is', 'program', 'for', 'splitting', 'word']
```

- **str.splitlines()** ทำการแยกกลุ่มคำออกจากสตริง โดยพิจารณาจาก \n ในกรณีที่กำหนดค่า num > 0 กลุ่มคำที่ถูกแยกออกจะมี \n ต่อท้าย

รูปแบบคำสั่ง: str.splitlines( num=string.count('\n'))

พารามิเตอร์: num > 0 กลุ่มคำที่ถูกแยกออกจะมี \n ต่อท้าย

ค่าที่ส่งกลับ: ข้อมูลชนิดลิสต์ของสตริงที่ถูกแยกแล้ว

ตัวอย่าง:

```
str = "Line1-a b c d e f\nLine2- a b c\n\nLine4- a b c d";
print (str.splitlines());
print (str.splitlines(0));
print (str.splitlines(3));
print (str.splitlines(4));
print (str.splitlines(5));
```


**OUTPUT**

```
['Line1-a b c d e f', 'Line2- a b c', '', 'Line4- a b c d']
['Line1-a b c d e f', 'Line2- a b c', '', 'Line4- a b c d']
['Line1-a b c d e f\n', 'Line2- a b c\n', '\n', 'Line4- a b c d']
['Line1-a b c d e f\n', 'Line2- a b c\n', '\n', 'Line4- a b c d']
['Line1-a b c d e f\n', 'Line2- a b c\n', '\n', 'Line4- a b c d']
```

- **str.startswith()** ตรวจสอบสตริงว่ามีข้อความที่ตรงกับข้อความที่ต้องการค้นหาหรือไม่

รูปแบบคำสั่ง: `str.startswith(str, beg=0,end=len(string))`

พารามิเตอร์: `str` เป็นข้อความที่ต้องการค้นหา, `beg` คือตำแหน่งเริ่มต้นที่ต้องการค้นหา, `end` คือตำแหน่งสุดท้ายของสตริงที่ต้องการค้นหา

ค่าที่ส่งกลับ: เป็นจริงเมื่อค้นหาพบ

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print (str.startswith('this'));
print (str.startswith('is', 2, 4));
print (str.startswith('this', 2, 4));
```


**OUTPUT**

```
True
True
False
```

- **str.strip()** ลบตัวอักษรที่ต้องการทิ้งทั้งจากด้านหน้าและด้านหลังสตริง โดยทำงานเหมือน lstrip() และ rstrip() ร่วมกัน

รูปแบบคำสั่ง: `str.strip([chars])`

พารามิเตอร์: `str` เป็นข้อความที่ต้องการลบตัวอักษร, `chars` ตัวอักษรที่ต้องการลบทิ้ง ถ้าไม่กำหนดตัวอักษรที่จะลบ ค่า default คือช่องว่าง

ค่าที่ส่งกลับ: สตริงที่ถูกลบตัวอักษรที่ต้องการออกแล้ว

ตัวอย่าง:

```
str = "0000000this is string example....wow!!!0000000";
print (str.strip('0')); #Remove 0 both sides
```


**OUTPUT**

```
this is string example....wow!!!
```

- **str.swapcase()** สลับตัวอักษรในสตริง เช่น ถ้าเดิมเป็นตัวเล็กจะสลับเป็นตัวใหญ่ แต่ถ้าเดิมเป็นตัวใหญ่จะสลับเป็นอักษรตัวเล็กแทน

รูปแบบคำสั่ง: `str.swapcase()`

พารามิเตอร์: `str` เป็นข้อความที่ต้องการสลับที่

ค่าที่ส่งกลับ: สตริงที่ถูกสลับที่ตัวอักษรแล้ว

ตัวอย่าง:

```
str = "This is string example....Wow!!!";
print (str.swapcase());
```

```
str = "THIS Is STRING EXAMPLE....WOW!!!";
print (str.swapcase());
```



THIS IS STRING EXAMPLE....wOW!!!
this iS string example....wOw!!!

**OUTPUT**

- **str.title()** แทนที่ตัวอักษรตัวแรกของกลุ่มคำในสตริงเป็นตัวอักษรตัวใหญ่ทั้งหมด

รูปแบบคำสั่ง: str.title()

พารามิเตอร์: str เป็นข้อความที่ต้องการเปลี่ยนตัวอักษรตัวแรกให้เป็นตัวใหญ่

ค่าที่ส่งกลับ: สตริงที่ถูกเปลี่ยนให้อักษรตัวแรกเป็นตัวใหญ่แล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print (str.title());
```



This Is String Example....Wow!!!

**OUTPUT**

- **str.upper()** แปลงอักษรในสตริงให้เป็นตัวใหญ่ทั้งหมด

รูปแบบคำสั่ง: str.upper()

พารามิเตอร์: str เป็นข้อความที่ต้องการเปลี่ยนเป็นอักษรตัวใหญ่

ค่าที่ส่งกลับ: สตริงที่ถูกเปลี่ยนให้เป็นอักษรตัวใหญ่แล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print ("str.capitalize() : ", str.capitalize())
```



str.upper() : THIS IS STRING EXAMPLE....WOW!!!

**OUTPUT**

- **str.zfill()** เติมอักษรเลข 0 ตามจำนวนที่กำหนด ตรงส่วนหน้าของสตริง

รูปแบบคำสั่ง: str.zfill(width)

พารามิเตอร์: str เป็นข้อความที่ต้องการเติมเลข 0, width คือความกว้างของสตริง

ค่าที่ส่งกลับ: สตริงที่ถูกเติมเลข 0 แล้ว

ตัวอย่าง:

```
str = "this is string example....wow!!!";
print ("Original string length = ", len(str)); #len = 32
chars
print (str.zfill(40)); # fill 0 from 32 to 40 chars
print (str.zfill(50)); # fill 0 from 32 to 50 chars
```



Original string length = 32
00000000this is string example....wOw!!!
00000000000000000000000000000000this is string example....wOw!!!

**OUTPUT**

- **str.isdecimal()** ตรวจสอบว่าตัวอักษรในสตริงเป็นเลขฐานสิบ

รูปแบบคำสั่ง: str.isdecimal()

พารามิเตอร์: str เป็นข้อความที่ต้องการตรวจสอบ

ค่าที่ส่งกลับ: เป็นจริงเมื่อตัวอักษรในสตริงเป็นเลขฐานสิบทั้งหมด

ตัวอย่าง:

```
str = u"This2009"; # u means Unicode
print (str.isdecimal());
str = u"23443434";
print (str.isdecimal());
```

|                                                                                             |               |
|---------------------------------------------------------------------------------------------|---------------|
| <br>OUTPUT | False<br>True |
|---------------------------------------------------------------------------------------------|---------------|

- **chr()** พิ้งชันแปลงจำนวนเต็มใดๆ เป็นตัวอักษร 1 ตัว

รูปแบบคำสั่ง: chr(i)

พารามิเตอร์: i คือจำนวนเต็มใดๆ มีค่าอยู่ระหว่าง 0 - 256

ค่าที่ส่งกลับ: สตริงที่เป็นตัวอักษรเพียง 1 ตัว

ตัวอย่าง:

```
a = 97 # integer of 'a' in ASCII code
b = 60 # integer of < in ASCII
print (chr(a));
print (chr(b));
```

|                                                                                               |        |
|-----------------------------------------------------------------------------------------------|--------|
| <br>OUTPUT | a<br>< |
|-----------------------------------------------------------------------------------------------|--------|

- **len()** พิ้งชันหาขนาดความยาวของอ็อปเจกต์ใดๆ

รูปแบบคำสั่ง: len(object)

พารามิเตอร์: object คืออ็อปเจกต์ใดๆ ที่ต้องการหาความยาว

ค่าที่ส่งกลับ: ความยาวที่เป็นเลขจำนวนเต็ม

ตัวอย่าง:

```
strs = 'Hello Python!'
lst = [1, 2, 3, 4]
tupl = (5, 6)
dicts = {1:'name',2:'Surname'}
print("len of string = ",len(strs))
print("len of list = ",len(lst));
print("len of tuple = ",len(tupl));
print("len of dictionary = ",len(dicts));
```


**OUTPUT**

```
len of string = 13
len of list = 4
len of tuple = 2
len of dictionary = 2
```

- **ord()** พังชันแปลงตัวอักษรใดๆ ให้เป็นตัวเลขจำนวนเต็ม

รูปแบบคำสั่ง: `ord(c)`

พารามิเตอร์: `c` คือตัวอักษรเพียง 1 ตัวเท่านั้น

ค่าที่ส่งกลับ: เลขจำนวนเต็มในรหัส ASCII

ตัวอย่าง:

```
print("Integer (ASCII Table) of a is ",ord('a'))
print("Integer (ASCII Table) of > is ",ord('>'))
print("Integer (ASCII Table) of # is ",ord('#'))
```


**OUTPUT**

```
Integer (ASCII Table) of a is 97
Integer (ASCII Table) of > is 62
Integer (ASCII Table) of # is 35
```

- **repr()** พังชันแปลงอ้อปเจกต์ใดๆ เป็นสตริง

รูปแบบคำสั่ง: `repr(object)`

พารามิเตอร์: `object` คืออ้อปเจกต์ใดๆ ที่ต้องการแปลงเป็นสตริง

ค่าที่ส่งกลับ: สตริง

ตัวอย่าง:

```
print("Convert Integer to String :",repr(1234))
print("Convert Float to String :",repr(12.34))
print("Convert List to String :",repr([1, 2, 3]))
print("Convert Tuple to String :",repr((4, 5)))
print("Convert Dictionary to String :",repr({1:'Python'}))
print("Convert Hex to String :",repr(0x345))
```


**OUTPUT**

```
Convert Integer to String : 1234
Convert Float to String : 12.34
Convert List to String : [1, 2, 3]
Convert Tuple to String : (4, 5)
Convert Dictionary to String : {1: 'Python'}
Convert Hex to String : 837
```

- **str()** พังชันแปลงอ้อปเจกต์ใดๆ เป็นสตริง เหมือน `repr()`

รูปแบบคำสั่ง: `str(object)`

พารามิเตอร์: `object` คืออ้อปเจกต์ใดๆ ที่ต้องการแปลงเป็นสตริง

ค่าที่ส่งกลับ: สตริง

ตัวอย่าง: เมื่อกับพังชัน `repr()` ทุกประการ

## 2. พังชันภายในสำหรับข้อมูลเชิงประกอบ (Built-in functions for composite data types)

## ฟังชัน และเมธอดสำหรับลิสต์ (Built-in list functions & methods)

### ฟังชันสำหรับลิสต์

- **len()** ฟังชันหาจำนวนของสมาชิก หรือความยาวในลิสต์

รูปแบบคำสั่ง: len(list)

พารามิเตอร์: list คือตัวแปรลิสต์ที่ต้องการหาจำนวนสมาชิก

ค่าที่ส่งกลับ: จำนวนของสมาชิกในลิสต์เป็นเลขจำนวนเต็ม

ตัวอย่าง:

```
list1, list2 = [123, 'xyz', 'zara'], [456, 'abc']
print ("First list length : ", len(list1));
print ("Second list length : ", len(list2));
```



First list length : 3  
Second list length : 2

**OUTPUT**

- **max()** ฟังชันค้นหาค่าข้อมูลที่มีค่าสูงที่สุดในลิสต์ ในกรณีที่เป็นตัวอักษรจะพิจารณาค่าตัวเลขประจำตัวอักษรนั้นๆ เช่น a = 97, b = 98 ในรหัส ASCII เป็นต้น

รูปแบบคำสั่ง: max(list)

พารามิเตอร์: list คือตัวแปรชนิดลิสต์

ค่าที่ส่งกลับ: สมาชิกในลิสต์ที่มีค่ามากที่สุด

ตัวอย่าง:

```
list1, list2 = ['123','xyz', 'zara', 'abc'], [456, 700, 200]
print ("Max value element : ", max(list1));
print ("Max value element : ", max(list2));
```



Max value element : zara  
Max value element : 700

**OUTPUT**

- **min()** ฟังชันค้นหาค่าข้อมูลที่มีค่าต่ำที่สุดในลิสต์

รูปแบบคำสั่ง: min(list)

พารามิเตอร์: list คือตัวแปรชนิดลิสต์

ค่าที่ส่งกลับ: สมาชิกในลิสต์ที่มีค่าน้อยที่สุด

ตัวอย่าง:

```
list1, list2 = ['123','xyz', 'zara', 'abc'], [456, 700, 200]
print ("Min value element : ", min(list1));
print ("Min value element : ", min(list2));
```



Min value element : 123  
Min value element : 200

**OUTPUT**

- **list()** พังชันแปลงข้อมูลชนิดทัพเพล (tuple) เป็นลิสต์ (list)

รูปแบบคำสั่ง: list(seq)

พารามิเตอร์: seq คือข้อมูลของตัวแปรชนิดทัพเพล

ค่าที่ส่งกลับ: ข้อมูลชนิดลิสต์

ตัวอย่าง:

```
aTuple = (123, 'xyz', 'zara', 'abc');
aList = list(aTuple)
print ("List elements : ", aList)
```



List elements : [123, 'xyz', 'zara', 'abc']

**OUTPUT**

- **any()** พังชันตรวจสอบหาว่าลิสต์ว่างหรือไม่

รูปแบบคำสั่ง: any(seq)

พารามิเตอร์: seq คือข้อมูลของตัวแปรชนิดลิสต์

ค่าที่ส่งกลับ: เป็นจริงเมื่อมีข้อมูลสมาชิกอย่างน้อย 1 ค่า หรือไม่ใช้ลิสต์ว่าง

ตัวอย่าง:

```
aList = []
print("Testing empty list : ", any(aList))
aList = [1, 2, 3]
print("Testing empty list : ", any(aList))
```



Testing empty list : False  
Testing empty list : True

**OUTPUT**

- **all()** พังชันตรวจสอบหาค่าที่เป็นเท็จของสมาชิก

รูปแบบคำสั่ง: all(seq)

พารามิเตอร์: seq คือข้อมูลของตัวแปรชนิดลิสต์

ค่าที่ส่งกลับ: เป็นจริงเมื่อข้อมูลสมาชิกในลิสต์ทั้งหมดไม่มีค่าใดเป็นค่าเท็จ

ตัวอย่าง:

```
aList = [4, 2, 5]
print("Testing false member:", all(aList))
aList = [4, 2, 0]
print("Testing false member:", all(aList))
aList = [4, 2, False]
print("Testing false member:", all(aList))
```



Testing false member: True  
Testing false member: False  
Testing false member: False

**OUTPUT**

- **range()** พังชันสร้างลำดับข้อมูลสมาชิกของลิสต์

|                |                                                                                                                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| รูปแบบคำสั่ง:  | <code>range([start], stop, [step])</code>                                                                                                                                                                                                       |
| พารามิเตอร์:   | start คือค่าเริ่มต้นของข้อมูล, stop คือค่าสิ้นสุดข้อมูล, step คือค่าที่ใช้ระบบถึงการสร้างข้อมูลในลักษณะเป็นช่วงๆ ถ้าไม่ไดระบุค่าใน step ข้อมูลจะเรียงในลักษณะจำนวนนับ คือ 1, 2, 3,...n และถ้าไม่กำหนดค่าใน start โปรแกรมจะถือว่าเริ่มจาก 0 เสมอ |
| ค่าที่ส่งกลับ: | ข้อมูลสมาชิกของตัวแปรลิสต์                                                                                                                                                                                                                      |
| ตัวอย่าง:      |                                                                                                                                                                                                                                                 |

```
lst1 = list(range(10))
lst2 = list(range(10, 20))
lst3 = list(range(0, 20, 2))
print("lst1 :", lst1)
print("lst2 :", lst2)
print("lst3 :", lst3)
```



**OUTPUT**

```
lst1 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
lst2 : [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
lst3 : [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

## เมธอดสำหรับลิสต์

- **list.append()** เพิ่มข้อมูลสมาชิกโดยการต่อท้ายลิสต์

รูปแบบคำสั่ง: `list.append(obj)`

พารามิเตอร์: list คือลิสต์ต้นฉบับ, obj คืออ็อปเจกต์ที่ต้องการเพิ่มต่อท้ายลิสต์

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("Before update list : ", aList);
aList.append(2009);
print ("After update list : ", aList);
```



**OUTPUT**

```
Before update list : [123, 'xyz', 'zara', 'abc']
After update list : [123, 'xyz', 'zara', 'abc', 2009]
```

- **list.count()** นับจำนวนสมาชิกที่ต้องการในลิสต์

รูปแบบคำสั่ง: `list.count(obj)`

พารามิเตอร์: list คือลิสต์ต้นฉบับ, obj คืออ็อปเจกต์ที่ต้องการนับ

ค่าที่ส่งกลับ: จำนวนอ็อปเจกต์ที่นับได้ มีค่าเป็นจำนวนเต็มบวก

ตัวอย่าง:

```
aList = [123, 'xyz', 'zara', 'abc', 123];
print ("Count for 123 : ", aList.count(123));
```

```
print ("Count for zara : ", aList.count('zara'));
```



**OUTPUT**

```
Count for 123 : 2
Count for zara : 1
```

- **list.extend()** เชื่อมสมาชิกของ 2 ลิสต์ได้ๆ เข้าด้วยกัน โดยเชื่อมต่อจากด้านหลัง

รูปแบบคำสั่ง: `list.extend(seq)`

พารามิเตอร์: `list` คือลิสต์ต้นฉบับ, `seq` คือลิสต์ใหม่ที่ต้องการเชื่อมต่อ

ค่าที่ส่งกลับ: `ไม่มี`

ตัวอย่าง:

```
aList = [123, 'xyz', 'zara', 'abc', 123];
bList = [2009, 'manni'];
aList.extend(bList)
print ("Extended List:", aList);
```



**OUTPUT**

```
Extended List:[123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']
```

- **list.index()** ค้นหาข้อมูลในลิสต์

รูปแบบคำสั่ง: `list.index(obj)`

พารามิเตอร์: `list` คือลิสต์ต้นฉบับ, `obj` คือข้อมูลที่ต้องการค้นหา

ค่าที่ส่งกลับ: ตำแหน่งของข้อมูลสมาชิกที่ค้นหาเจอในอันดับต้นของลิสต์ ถ้าค้นหาไม่พบจะเกิด exception error

ตัวอย่าง:

```
aList = [125, 'xyz', 'zara', 'abc', 123];
print ("Index for 123 : ", aList.index(123));
print ("Index for zara : ", aList.index('zara'));
```



**OUTPUT**

```
Index for 123 : 4
Index for zara : 2
```

- **list.insert()** เพิ่มข้อมูลสมาชิกใหม่เข้าไปในลิสต์

รูปแบบคำสั่ง: `list.insert(index, obj)`

พารามิเตอร์: `list` คือลิสต์ต้นฉบับ, `index` คือตำแหน่งที่ต้องการเพิ่มสมาชิก, `obj` คือข้อมูลสมาชิกใหม่ที่ต้องการเพิ่ม

ค่าที่ส่งกลับ: `ไม่มี`

ตัวอย่าง:

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert(3, 2009)
print ("Insert 2009 to list : ", aList)
aList.insert(-1,123)
```

```
print ("Insert 123 before end list : ", aList)
```



**OUTPUT**

```
Insert 2009 to list : [123, 'xyz', 'zara', 2009, 'abc']
Insert 123 before end list:[123,'xyz','zara',2009,123,'abc']
```

- **list.pop()** ลบข้อมูลสมาชิกออกจากลิสต์ โดย default จะลบสมาชิกที่อยู่ในตำแหน่งท้ายสุด ของลิสต์ออกก่อน แต่ถ้าต้องการลบข้อมูลสมาชิกตำแหน่งใดๆ ในลิสต์ออก ให้กำหนดชื่อ สมาชิกที่ต้องการลบใน obj

รูปแบบคำสั่ง: `list.pop(obj=list[-1])`

พารามิเตอร์: list คือลิสต์ต้นฉบับ, obj คือข้อมูลสมาชิกที่ต้องการลบออก

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("Before pop list : ", aList);
print ("1'st pop (last list): ", aList.pop());
print ("After 1'st pop list : ", aList);
print ("2'nd pop [index 1]: ", aList.pop(1));
print ("After 2'nd pop list : ", aList);
```



**OUTPUT**

```
Before pop list : [123, 'xyz', 'zara', 'abc']
1'st pop (last list): abc
After 1'st pop list : [123, 'xyz', 'zara']
2'nd pop [index 1]: xyz
After 2'nd pop list : [123, 'zara']
```

- **list.remove()** ลบข้อมูลสมาชิกออกจากลิสต์ โดยระบุชื่อของสมาชิกที่ต้องการลบ

รูปแบบคำสั่ง: `list.remove(obj)`

พารามิเตอร์: list คือลิสต์ต้นฉบับ, obj คือชื่อสมาชิกที่ต้องการลบออก

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
List = [123, 'xyz', 'zara', 'abc', 'xyz'];
print ("Before remove list : ", aList);
aList.remove('xyz');
print ("Remove xyz member : ", aList);
aList.remove('abc');
print ("Remove abc member : ", aList);
```



**OUTPUT**

```
Before remove list : [123, 'xyz', 'zara', 'abc', 'xyz']
Remove xyz member : [123, 'zara', 'abc', 'xyz']
Remove abc member : [123, 'zara', 'xyz']
```

- **list.reverse()** จัดเรียงข้อมูลของสมาชิกภายในลิสต์ แบบย้อนกลับ หรือสลับทิศทางกัน

รูปแบบคำสั่ง: `list.reverse()`

พารามิเตอร์: list คือลิสต์ต้นฉบับ

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
aList = [1, 2, 3, 4, 5, 6, 7];
print ("Before reverse list : ", aList)
aList.reverse();
print ("After reverse list : ", aList)
```



OUTPUT

```
Before reverse list : [1, 2, 3, 4, 5, 6, 7]
After reverse list : [7, 6, 5, 4, 3, 2, 1]
```

- **list.sort()** จัดเรียงข้อมูลของสมาชิกภายในลิสต์ โดยพิจารณาจากค่าประจำตำแหน่งของตัวอักษรแต่ละตัว

รูปแบบคำสั่ง: `list.sort([func])`

พารามิเตอร์: `list` คือลิสต์ต้นฉบับ, `func` คือวิธีที่ใช้จัดเรียงข้อมูล จะมีหรือไม่มีก็ได้

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
aList = ['123', 'xyz', 'zara', 'abc', 'xyz'];
bList = [1, 5, 7, 4, 5, 6, 3]
aList.sort();
bList.sort();
print ("Sorting aList : ", aList);
print ("Sorting bList : ", bList);
```



OUTPUT

```
Sorting aList : ['123', 'abc', 'xyz', 'xyz', 'zara']
Sorting bList : [1, 3, 4, 5, 5, 6, 7]
```

## ฟังชันภายในสำหรับทัพเพิล (Built-in tuple functions)

- **len()** ฟังชันนับจำนวนสมาชิกในทัพเพิล

รูปแบบคำสั่ง: `len(tuple)`

พารามิเตอร์: `tuple` คือทัพเพิลต้นฉบับ

ค่าที่ส่งกลับ: จำนวนของสมาชิกเป็นจำนวนเต็มบวก

ตัวอย่าง:

```
tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')
print ("First tuple length : ", len(tuple1));
print ("Second tuple length : ", len(tuple2));
```



OUTPUT

```
First tuple length : 3
Second tuple length : 2
```

- **max()** ฟังชันค้นหาสมาชิกที่มีค่ามากที่สุดในทัพเพิล

- รูปแบบคำสั่ง: max(tuple)  
 พารามิเตอร์: tuple คือทัพเพิลตันฉบับ  
 ค่าที่ส่งกลับ: สมาชิกที่มีค่ามากที่สุดในทัพเพิล

ตัวอย่าง:

```
tuple1, tuple2 = ('123', 'xyz', 'zara', 'abc'), (456, 700, 200)
print ("Max value element : ", max(tuple1));
print ("Max value element : ", max(tuple2));
```



Max value element : zara  
 Max value element : 700

**OUTPUT**

- **min()** ค้นหาสมาชิกที่มีค่าน้อยที่สุดในทัพเพิล

- รูปแบบคำสั่ง: min(tuple)  
 พารามิเตอร์: tuple คือทัพเพิลตันฉบับ  
 ค่าที่ส่งกลับ: สมาชิกที่มีค่าน้อยที่สุดในทัพเพิล

ตัวอย่าง:

```
tuple1, tuple2 = ('123', 'xyz', 'zara', 'abc'), (456, 700, 200)
print ("Min value element : ", min(tuple1));
print ("Min value element : ", min(tuple2));
```



Min value element : 123  
 Min value element : 200

**OUTPUT**

- **tuple()** แปลงข้อมูลชนิดลิสต์ไปเป็นทัพเพิล

- รูปแบบคำสั่ง: tuple(seq)  
 พารามิเตอร์: seq คือข้อมูลของตัวแปรชนิดลิสต์  
 ค่าที่ส่งกลับ: ข้อมูลชนิดทัพเพิล

ตัวอย่าง:

```
aList = (123, 'xyz', 'zara', 'abc');
aTuple = tuple(aList)
print ("Tuple elements : ", aTuple)
```



Tuple elements : (123, 'xyz', 'zara', 'abc')

**OUTPUT**

## ฟังชัน และเมธอดสำหรับดิกชันนารี (Built-in Dictionary Functions & Methods)

### ฟังชันสำหรับดิกชันนารี (Dictionary Functions)

- **len()** ฟังชันหาจำนวนของสมาชิก หรือความยาวในดิกชันนารี

- รูปแบบคำสั่ง: len(dict)
- พารามิเตอร์: dict คือตัวแปรชนิดดิกชันนารี
- ค่าที่ส่งกลับ: จำนวนของสมาชิกในดิกชันนารี
- ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7};
print ("Length : %d" % len (dict))
```



Length : 2

- **str()** แสดงโครงสร้างของดิกชันนารีทั้งหมดในรูปของสตริง

- รูปแบบคำสั่ง: str(dict)
- พารามิเตอร์: dict คือตัวแปรชนิดดิกชันนารี
- ค่าที่ส่งกลับ: สตริงที่แสดงโครงสร้างของดิกชันนารี

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7};
print ("Equivalent String : %s" % str (dict))
```



Equivalent String : {'Age': 7, 'Name': 'Zara'}

- **type()** แสดงชนิดของตัวแปร

- รูปแบบคำสั่ง: type(variable)
- พารามิเตอร์: variable คืออ้อปเจ็กต์ใดๆ
- ค่าที่ส่งกลับ: สตริงแสดงชนิดของตัวแปร

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7};
print ("Variable Type : %s" % type (dict))
```



Variable Type : <class 'dict'>

## เมธอดสำหรับดิกชันนารี (Dictionary methods)

- **dict.clear()** เคลียร์ข้อมูลทั้งหมดออกจากดิกชันนารี

- รูปแบบคำสั่ง: dict.clear()
- พารามิเตอร์: dict คือตัวแปรดิกชันนารี
- ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7};
print ("Start Len : %d" % len(dict))
dict.clear()
print ("End Len : %d" % len(dict))
```



Start Len : 2  
End Len : 0

**OUTPUT**

- **dict.copy()** สำเนาข้อมูลของ딕셔너리

รูปแบบคำสั่ง: dict.copy()

พารามิเตอร์: dict คือตัวแปร딕셔너รี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
dict1 = { 'Name': 'Zara', 'Age': 7};
dict2 = dict1.copy()
print ("New Dictionary : %s" % str(dict2))
```



New Dictionary : { 'Name': 'Zara', 'Age': 7}

**OUTPUT**

- **dict.fromkeys()** สร้าง딕셔너รีใหม่ โดยการสำเนาคีย์มาจาก딕셔너รีเดิม หรือกำหนดค่าเริ่มต้นให้กับข้อมูล

รูปแบบคำสั่ง: dict.fromkeys(seq[, value]))

พารามิเตอร์: seq คือ딕셔너รีเดิมที่ต้องการสำเนาคีย์, value คือค่าเริ่มต้นที่ต้องการกำหนดให้กับตัวแปร딕셔너รีที่สร้างขึ้นใหม่

ค่าที่ส่งกลับ: 딕셔너รีใหม่ที่สร้างขึ้น

ตัวอย่าง:

```
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq) #new dict & copy keys
print ("New Dictionary : %s" % str(dict))
dict = dict.fromkeys(seq, 10) #new dict by copy keys & set value
print ("New Dictionary : %s" % str(dict)))
```



New Dictionary : { 'sex': None, 'name': None, 'age': None}
New Dictionary : { 'sex': 10, 'name': 10, 'age': 10}

**OUTPUT**

- **dict.get()** ค้นหาข้อมูลใน딕셔너รี โดยใช้คีย์ในการค้นหา

รูปแบบคำสั่ง: dict.get(key, default=None)

พารามิเตอร์: key คือคีย์ที่ใช้ค้นหา, default ระบุค่าที่ต้องการให้ส่งกลับ

ค่าที่ส่งกลับ: แสดงข้อมูลที่ค้นพบ แต่ถ้าไม่พบจะคืนค่าเป็น None

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.get('Age'))
print ("Value : %s" % dict.get('Sex', "Never"))
```



**OUTPUT**

Value : 27  
Value : Never

- **dict.items()** แสดงคู่ของคีย์และข้อมูล

รูปแบบคำสั่ง: dict.items()

พารามิเตอร์: dict คือข้อมูลชนิดดิกชันนารี

ค่าที่ส่งกลับ: ลิสต์ที่บรรจุคู่ของคีย์และข้อมูล

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7}
print ("Value : %s" % dict.items())
```



**OUTPUT**

Value : dict\_items([('Age', 7), ('Name', 'Zara')])

- **dict.keys()** แสดงรายการของคีย์ทั้งหมดในดิกชันนารี

รูปแบบคำสั่ง: dict.keys()

พารามิเตอร์: dict คือข้อมูลชนิดดิกชันนารี

ค่าที่ส่งกลับ: รายการของคีย์ทั้งหมด

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7}
print ("Value : %s" % dict.keys())
```



**OUTPUT**

Value : dict\_keys(['Name', 'Age'])

- **dict.setdefault()** ทำงานคล้าย dict.get แต่แตกต่างที่เมื่อค้นหาคีย์ใดๆ ในดิกชันนารีไม่เจอ

จะแทนที่คีย์และข้อมูลใหม่ใส่เข้าไปเป็นสมาชิกใหม่ทันที

รูปแบบคำสั่ง: dict.setdefault(key, default=None)

พารามิเตอร์: dict คือข้อมูลชนิดดิกชันนารี, key คือคีย์ที่ต้องการค้นหา, default คือค่าที่ต้องการทำหนدให้เมื่อค้นหาข้อมูลใดๆ ไม่พบ ถ้าไม่กำหนดจะเป็น None

ค่าที่ส่งกลับ: รายการของข้อมูลทั้งหมด

ตัวอย่าง:

```
dict = { 'Name': 'Zara', 'Age': 7}
```

```

print ("Value : %s" % dict.setdefault('Age', None))
print ("Value : %s" % dict.setdefault('Sex', None))
print ("Value : %s" % dict.setdefault('ID', '001'))
print (dict)

```



**OUTPUT**

```

Value : 7
Value : None
Value : 001
{'Age': 7, 'Sex': None, 'Name': 'Zara', 'ID': '001'}

```

- **dict.update()** ปรับปรุง หรือเชื่อมข้อมูลใน딕ชันนารี 2 ดิกชันนารีได้ๆ เช่น

รูปแบบคำสั่ง: `dict.update(dict2)`

พารามิเตอร์: `dict` คือข้อมูลชนิดดิกชันนารี, `dict2` คือข้อมูลดิกชันนารีที่ต้องการเชื่อมต่อเข้ากับ `dict`

ค่าที่ส่งกลับ: `ไม่มี`

ตัวอย่าง:

```

dict = {'Name': 'Zara', 'Age': 7}
dict2 = {'Sex': 'female' }
dict.update(dict2)
print ("Value : %s" % dict)

```



**OUTPUT**

```

Value : {'Sex': 'female', 'Age': 7, 'Name': 'Zara'}

```

- **dict.values()** แสดงรายการข้อมูลในดิกชันนารีทั้งหมด

รูปแบบคำสั่ง: `dict.values()`

พารามิเตอร์: `dict` คือข้อมูลชนิดดิกชันนารี

ค่าที่ส่งกลับ: `ลิสต์ของรายการข้อมูล (values) ในดิกชันนารีทั้งหมด`

ตัวอย่าง:

```

dict = {'Name': 'Zara', 'Age': 7}
print ("Value : %s" % dict.values())

```



**OUTPUT**

```

Value : dict_values(['Zara', 7])

```

- **dict.has\_key(key)** ค้นหาว่าคีย์อยู่ในดิกชันนารีหรือไม่ จะไม่มีใน Python 3.0 แล้ว แต่

ผู้เขียนโปรแกรมสามารถใช้ `in` แทนได้ดังนี้

```

>>> d = {'a': 1, 'b': 2}
>>> check = 'a' in d
>>> print(check)
True

```

## ฟังชัน และเมธอดสำหรับเซต (Set built-in functions and methods)

### ฟังชันสำหรับเซต

- **len()** คำนวณจำนวนสมาชิกทั้งหมดที่อยู่ในเซต

รูปแบบคำสั่ง: len(object)

พารามิเตอร์: object คือข้อมูลชนิดเซต

ค่าที่ส่งกลับ: จำนวนสมาชิกทั้งหมดที่อยู่ในเซต

ตัวอย่าง:

```
setx = {1, 2, 3, 4, 5}
print("Number of set :", len(setx))
```



Number of set : 5

**OUTPUT**

- **max()** ค้นหาสมาชิกที่มีค่ามากที่สุดในเซต

รูปแบบคำสั่ง: max(set)

พารามิเตอร์: set คือข้อมูลชนิดเซต

ค่าที่ส่งกลับ: สมาชิกที่มีค่ามากที่สุดในเซต

ตัวอย่าง:

```
setx = {1, 2, 3, 4, 5, 6, 7}
print("The largest item of set :", max(setx))
```



The largest item of set : 7

**OUTPUT**

- **min()** ค้นหาสมาชิกที่มีค่าน้อยที่สุดในเซต

รูปแบบคำสั่ง: min(set)

พารามิเตอร์: set คือข้อมูลชนิดเซต

ค่าที่ส่งกลับ: สมาชิกที่มีค่าน้อยที่สุดในเซต

ตัวอย่าง:

```
setx = {1, 2, 3, 4, 5, 6, 7}
print("The smallest item of set :", min(setx))
```



The smallest item of set : 1

**OUTPUT**

### เมธอดสำหรับเซต

- **s.clear()** เคลียร์ข้อมูลทั้งหมดในเซต

รูปแบบคำสั่ง: s.clear()

พารามิเตอร์: s คือข้อมูลชนิดเซต

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
print("Before set was cleared :",s)
s.clear()
print("After set was cleared :",s)
```



Before set was cleared : {1, 2, 3, 4, 5, 6, 7}  
After set was cleared : set()

**OUTPUT**

- **s.copy()** การทำสำเนาเซต

รูปแบบคำสั่ง: s.copy()

พารามิเตอร์: s คือข้อมูลชนิดเซต

ค่าที่ส่งกลับ: เซตใหม่ที่สำเนามาจากเซต s

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
newset = s.copy()
print("New set after coopy :",newset)
```



New set after coopy : {1, 2, 3, 4, 5, 6, 7}

**OUTPUT**

- **s.pop()** การลบสมาชิกตัวแรกออกจากเซต

รูปแบบคำสั่ง: s.pop()

พารามิเตอร์: s คือข้อมูลชนิดเซต

ค่าที่ส่งกลับ: ไม่มี แต่ถ้าเซตว่างจะเกิด error

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
print("After pop set :",s.pop())
print("remained item in set :",s)
```



After pop set : 1  
remained item in set : {2, 3, 4, 5, 6, 7}

**OUTPUT**

- **s.add()** เพิ่มสมาชิกเข้าไปในเซต

รูปแบบคำสั่ง: s.add(new)

พารามิเตอร์: s คือข้อมูลชนิดเซต, new คือข้อมูลใหม่ที่ต้องการเพิ่มเข้าไปในเซต

ค่าที่ส่งกลับ: ไม่มี แต่ถ้าเช็คว่างจะเกิด error

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
s.add(8)
print("After insert item into set :", s)
```



**OUTPUT**

After insert item into set : {1, 2, 3, 4, 5, 6, 7, 8}

- **s.remove()** ลบสมาชิกตัวที่ต้องการออกจากเซต

รูปแบบคำสั่ง: s.remove(old)

พารามิเตอร์: s คือข้อมูลชนิดเซต, old คือข้อมูลที่ต้องการลบออกจากเซต

ค่าที่ส่งกลับ: ไม่มี แต่ถ้าเช็คว่างจะเกิด error

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
s.remove(5)
print("After delete item from set :", s)
```



**OUTPUT**

After delete item from set : {1, 2, 3, 4, 6, 7}

- **s.discard()** ลบสมาชิกที่ต้องการออกจากเซต (ไม่แสดง error ถ้าไม่พบสมาชิกที่จะลบ)

รูปแบบคำสั่ง: s.discard(old)

พารามิเตอร์: s คือข้อมูลชนิดเซต, old คือข้อมูลที่ต้องการลบออกจากเซต

ค่าที่ส่งกลับ: ไม่มี แต่ถ้าเช็คว่าง จะไม่เกิด error เมื่อ non remove

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
s.discard(5)
print("After delete item from set :", s)
s.discard(5)
```



**OUTPUT**

After delete item from set : {1, 2, 3, 4, 6, 7}

- **s.update()** ทำการรวม 2 เซตใดๆ เข้าด้วยกัน

รูปแบบคำสั่ง: s.update(new)

พารามิเตอร์: s คือข้อมูลชนิดเซต, new คือเซตใหม่ที่ต้องการรวมกับ s

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
s1 = {4, 5, 8, 9}
```

```
s.update(s1)
print("After update set :",s)
```



**OUTPUT**

After update set : {1, 2, 3, 4, 5, 6, 7, 8, 9}

- **s.intersection\_update()** intersection ระหว่าง 2 เซตได้

รูปแบบคำสั่ง: s.intersection\_update(new)

พารามิเตอร์: s คือข้อมูลชนิดเซต, new คือเซตใหม่ที่ต้องการทำ intersection

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
s1 = {4, 5, 8, 9}
s.intersection_update(s1)
print("After intersect set :",s)
```



**OUTPUT**

After intersect set : {4, 5}

- **s.difference\_update()** ลบสมาชิกที่เหมือนกันออกจาก s

รูปแบบคำสั่ง: s.difference\_update(new)

พารามิเตอร์: s คือข้อมูลชนิดเซต, new คือเซตที่ใช้ในการเปรียบเทียบ

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
new = {4, 5, 8, 9}
s.difference_update(new)
print("After difference update set :",s)
```



**OUTPUT**

After difference update set : {1, 2, 3, 6, 7}

- **s.issubset()** ตรวจสอบความเป็น subset

รูปแบบคำสั่ง: s.issubset(set)

พารามิเตอร์: s คือข้อมูลชนิดเซต, set คือเซตที่ใช้ในการเปรียบเทียบ

ค่าที่ส่งกลับ: เป็นจริง ถ้า s เป็น subset ของ set ถ้าไม่ใช่จะเป็นเท็จ

ตัวอย่าง:

```
set = {1, 2, 3, 4, 5, 6, 7}
s = {3,4}
print("After check subset :",s.issubset(set))
```



After check subset : True

**OUTPUT**

- **s.union()** union ระหว่าง s และ new

รูปแบบคำสั่ง: `s.union(new)`

พารามิเตอร์: s คือชื่อชุดข้อมูลชนิดเซต, new คือเซตที่ใช้ดำเนินการ union

ค่าที่ส่งกลับ: เซตที่ผ่านการดำเนินการ union มาแล้ว

ตัวอย่าง:

```
s = {1, 2, 3, 4, 5, 6, 7}
new = {4, 5, 8, 9}
s1 = s.union(new)
print("After union set :", s1)
```



After union set : {1, 2, 3, 4, 5, 6, 7, 8, 9}

**OUTPUT**

### 3. เมธอดเกี่ยวกับการบริหารจัดการแฟ้มข้อมูลและไดเรคทรอรี่ (File & Directory Methods)

การใช้งานเมธอดเกี่ยวกับแฟ้มข้อมูลจำเป็นต้องเปิดแฟ้มด้วยเมธอด `open()` และใช้ออปเจก (File object) ที่ได้จากเมธอด `open` เพื่ออ้างอิงถึงแฟ้มจริงในระบบ ทั้งนี้การใช้เมธอดใดๆ ต้องขึ้นอยู่กับโหมดที่กำหนดในเมธอด `open` ด้วย เช่น ถ้าต้องการเขียนแฟ้มแบบต่อเนื่องจะต้องเปิดโดยใช้โหมด `a+` เป็นต้น

- **file.close():** ปิดแฟ้มที่เปิดใช้งานอยู่ เมื่อแฟ้มถูกปิดลงแล้วจะไม่สามารถดำเนินการใดๆ กับแฟ้มได้อีก ถ้ามีการอ่านแฟ้มจะเกิด `ValueError`, ขณะเปิดแฟ้ม A ข้างไว้ และเปิดแฟ้ม B อีก โดยใช้ตัวแปรที่อ้างอิงตัวเดียวกัน แฟ้ม A จะถูกปิดโดยอัตโนมัติ

รูปแบบคำสั่ง: `file.close()`

พารามิเตอร์: ไม่มี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
Open a file
file = open("test.txt", "wb")
print ("Name of the file: ", file.name)
Close opened file
file.close()
```

ผลลัพธ์:



Name of the file: test.txt

**OUTPUT**

- **file.flush()**: เขียนข้อมูลในหน่วยความจำลงแฟ้ม เมื่อทำการปิดแฟ้ม ข้อมูลที่อยู่ในหน่วยความจำจะถูกเขียนลงแฟ้มก่อนแฟ้มถูกปิด แต่ผู้เขียนโปรแกรมสามารถสั่งให้เขียนข้อมูลลงแฟ้มก่อนได้ ด้วยเมธอด flush

รูปแบบคำสั่ง: **file.flush()**

พารามิเตอร์: ไม่มี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
Open a file
file = open("test.txt", "wb")
print ("Name of the file: ", file.name)
flush operation
file.flush()
Close opened file
file.close()
```

เอกสารพูด:



Name of the file: test.txt

**OUTPUT**

- **file.fileno()**: แสดงค่าหมายเลขอ้างอิงแฟ้มจริง โดยปกติระบบปฏิบัติการจะอ้างอิงแฟ้มใดๆ จะใช้ค่าจำนวนเต็มในการอ้างอิง เรียกว่า integer file descriptor เมธอด fileno() จะส่งค่าตัวเลขดังกล่าวให้กับผู้เรียกใช้งาน

รูปแบบคำสั่ง: **file.fileno()**

พารามิเตอร์: ไม่มี

ค่าที่ส่งกลับ: ตัวเลขจำนวนเต็มมาก

ตัวอย่าง:

```
Open a file
file = open("test.txt", "wb")
print ("Name of the file: ", file.name)
fid = file.fileno()
print ("File Descriptor: ", fid)
Close opened file
file.close()
```

เอกสารพูด:



Name of the file: test.txt  
File Descriptor: 3

OUTPUT

- **file.isatty():** เชื่อมต่อแฟ้มกับเทอร์มินอล (tty) ในระบบปฏิบัติการยูนิกซ์หรือลินุกซ์ จะมองว่าโครงสร้างแฟ้มเป็นอุปกรณ์หนึ่ง ๆ ที่เชื่อมต่อกับระบบปฏิบัติการ เมื่อต้องการใช้งานแฟ้มจำเป็นต้องสั่งเชื่อมโครงสร้างเข้ากับโครงสร้างแฟ้มของระบบ (mount) การเข้าถึงแฟ้มในยูนิกซ์จะผ่านโปรแกรมตัวหนึ่ง บางครั้งเรียกเทอร์มินอล (Terminal) เมื่อแฟ้มดังกล่าวถูกเปิดด้วยเทอร์มินอล เรียกว่าการเชื่อม (connect)

รูปแบบคำสั่ง: **file.fileno()**

พารามิเตอร์: "ไม่มี"

ค่าที่ส่งกลับ: เป็นจริงเมื่อแฟ้มดังกล่าวเชื่อมกับเทอร์มินอล กรณีอื่นๆ จะเป็นเท็จ  
ตัวอย่าง:

```
status = file.isatty()
print ("Return value : ", status)
```

เอกสาร์พุต:



Return value : False

OUTPUT

- **next():** ทำหน้าที่อ่านข้อมูลจากแฟ้มเข้ามาทำงานครั้งละ 1 บรรทัด จนกว่าจะหมดแฟ้ม นิยมใช้งานควบคู่กับคำสั่งวนซ้ำ คือ for

หมายเหตุ: file.next() ถูกยกเลิกการใช้งานใน Python 3

รูปแบบคำสั่ง: **next(file)**

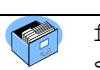
พารามิเตอร์: "ไม่มี"

ค่าที่ส่งกลับ: ข้อความบรรทัดถัดไป

ตัวอย่าง:

```
file = open("INPUT.txt", 'r+')
for index in range(5):
 line = next(file)
 print ("Line No %d - %s" % (index, line))
file.close()
```

อินพุต:



first line  
second line  
third line  
forth line  
fifth line

เอกสาร์พุต:


**OUTPUT**

Line No 0 - first line  
 Line No 1 - second line  
 Line No 2 - third line  
 Line No 3 - forth line  
 Line No 4 - fifth line

- **file.read():** ทำหน้าที่อ่านข้อมูลจากแฟ้มทั้งหมด

รูปแบบคำสั่ง: **file.read(size)**

พารามิเตอร์: size ขนาดของข้อมูลที่ต้องการอ่านจากแฟ้ม มีหน่วยเป็นไบต์

ค่าที่ส่งกลับ: ข้อมูลที่อ่านจากแฟ้ม มีหน่วยเป็นไบต์

ตัวอย่าง:

```
file = open("INPUT.txt", 'r+')
line = file.read(10)
print ("Read Line: %s" % (line))
file.close()
```

เอกสารพุต:


**OUTPUT**

Read Line: first line

- **file.readline():** ทำหน้าที่อ่านข้อมูลจากแฟ้มทีละบรรทัด

รูปแบบคำสั่ง: **file.readline([size])**

พารามิเตอร์: size ขนาดของข้อมูลที่ต้องการอ่านจากแฟ้ม มีหน่วยเป็นไบต์ เมื่อไม่

กำหนดจะอ่านทีละบรรทัดจนหมดแฟ้ม

ค่าที่ส่งกลับ: ข้อมูลที่อ่านจากแฟ้ม ครั้งละ 1 บรรทัด

ตัวอย่าง:

```
file = open("INPUT.txt", 'r+')
line = file.readline()
print ("Read Line: %s" % (line))
line = file.readline(5)
print ("Read Line: %s" % (line))
file.close()
```

เอกสารพุต:


**OUTPUT**

Read Line: first line

Read Line: secon

- **file.readlines():** ทำหน้าที่อ่านข้อมูลจากแฟ้มครั้งละหลายบรรทัด

รูปแบบคำสั่ง: **file.readlines([sizehint])**

พารามิเตอร์: sizehint ขนาดของข้อมูลที่ต้องการอ่านจากแฟ้ม มีหน่วยเป็นไบต์ เมื่อกำหนดค่า sizehint จำนวนบรรทัดจะถูกคำนวณจากค่า sizehint แทนการอ่านแฟ้มทั้งแฟ้ม

ค่าที่ส่งกลับ: ข้อมูลที่อ่านจากแฟ้มครั้งละหลายบรรทัด แต่ละบรรทัดจะปิดด้วย \n  
ตัวอย่าง:

```
file = open("INPUT.txt", 'r+')
line = file.readlines()
print ("Read Line: %s" % (line))
line = file.readlines(2)
print ("Read Line: %s" % (line))
file.close()
```

ผลลัพธ์:



**OUTPUT**

```
Read Line: ['first line\n', 'second line\n', 'third
line\n', 'forth line\n', 'fifth line']
Read Line: []
```

- **file.seek():** เลื่อนตำแหน่งตัวชี้ในแฟ้มตามค่าที่ระบุใน offset

รูปแบบคำสั่ง: file.seek(offset[, whence])

พารามิเตอร์: offset เป็นตำแหน่งที่เริ่มอ่าน-เขียนแฟ้มข้อมูล, whence (option) เป็นการกำหนดจุดอ้างอิงในการอ่าน-เขียนแฟ้ม โดยปกติจะเป็น 0 หมายถึงอ้างอิงจากต้นแฟ้ม ถ้ากำหนดเป็น 1 คืออ้างอิงจากตำแหน่งปัจจุบัน และถ้าเป็น 2 หมายถึงอ้างกับตำแหน่งจุดจบของแฟ้ม

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
file = open("INPUT.txt", 'r+')
line = file.readline()
print ("Read Line: %s" % (line))

Again set the pointer to the beginning
file.seek(0, 0)
line = file.readline()
print ("Read Line: %s" % (line))
Again set the pointer to the current position
file.seek(0, 1)
line = file.readline()
print ("Read Line: %s" % (line))
file.close()
```

ผลลัพธ์:



**Input File: TEST.txt**

```
first line
second line
third line
forth line
fifth line
```

ເອົາຕີ່ພຸດ:



**OUTPUT**

```
Read Line: first line
Read Line: first line
Read Line: second line
```

- **file.tell():** ຕຽບສອບຕຳແໜ່ງຕົວຊີ້ແພັນປັຈຈຸບັນ

ຮູບແບບຄຳສັ່ງ: `file.tell()`

ພາຣາມີເຕອຮີ: "ໄມ່ມີ"

ຄໍາທີ່ສັງກລັບ: ຕຳແໜ່ງຂອງຕົວຊີ້ແພັນທີ່ກຳລັງອ່ານ-ເຂີຍໃນປັຈຈຸບັນ

ຕົວອ່າງ:

```
file = open("INPUT.txt", 'r+')
line = file.readline()
print ("Read Line: %s" % (line))
Get the current position of the file.
pos = file.tell()
print ("Current Position: %d" % (pos))
file.close()
```

ເອົາຕີ່ພຸດ:



**OUTPUT**

```
Read Line: first line
Current Position: 12
```

- **file.write():** ເຂີຍຂ້ອມໝູລອງແພັນ

ຮູບແບບຄຳສັ່ງ: `file.write(str)`

ພາຣາມີເຕອຮີ: ຂ້ອຄວາມຫົ່ວ່າສຕຣິງທີ່ຕ້ອງການເຂີຍ

ຄໍາທີ່ສັງກລັບ: "ໄມ່ມີ"

ຕົວອ່າງ:

```
file = open("INPUT.txt", 'a+')
str = "\nsixth line"
Write a line at the end of the file.
file.seek(0, 2)
line = fo.write(str)

file = open("INPUT.txt", "r+")
Now read complete file from beginning.
file.seek(0,0)
for index in range(6):
 line = next(file)
```

```
 print ("Line No %d - %s" % (index, line))
file.close()
```

ເອກົດ:



**OUTPUT**

```
Line No 0 - first line
Line No 1 - second line
Line No 2 - third line
Line No 3 - forth line
Line No 4 - fifth line
Line No 5 - sixth line
```

- **file.writelines():** ເຂີນຂໍ້ມູນລັງແພັ່ນຄັ້ງລະຫວາຍບຣທັດ

ຮູບແບບຄຳສັ່ງ: **file.writelines(sequence)**

ພາຣາມີເຕືອຮ່: ຂັດຂອງຂໍ້ຄວາມໂຮງສຕຣິງ ທີ່ຕ້ອງການເຂີນ

ຄ່າທີ່ສັ່ງກັບ: ໄນມີ

ຕົວຢ່າງ:

```
file = open("INPUT.txt", 'a+')
seq = ["This is 6th line\n", "This is 7th line"]
Write sequence of lines at the end of the file.
file.seek(0, 2)
line = file.writelines(seq)

Now read complete file from beginning.
file.seek(0,0)
for index in range(7):
 line = next(file)
 print ("Line No %d - %s" % (index, line))
file.close()
```

ເອກົດ:



**OUTPUT**

```
Line No 0 - first line
Line No 1 - second line
Line No 2 - third line
Line No 3 - forth line
Line No 4 - fifth line
Line No 5 - This is 6th line
Line No 6 - This is 7th line
```

- **os.access():** ຖດສອບສິທິທິນີການເຂົ້າສົ່ງແພັ່ນ ໂດຍໃຊ້ GID/UID

ຮູບແບບຄຳສັ່ງ: **os.access(path, mode)**

ພາຣາມີເຕືອຮ່: path ອີ່ມີຂອງແພັ່ນຂໍ້ມູນທີ່ຕ້ອງການປັບປຸງ

Mode ອີ່ມີຄຸນສົມບັດທີ່ໃຊ້ສໍາຮັບເຂົ້າສົ່ງແພັ່ນ ປະກອບດ້ວຍ

**os.F\_OK** ອີ່ມີການປັບປຸງແພັ່ນດັ່ງກ່າວມີອຸ່ງຈິງ

**os.R\_OK** ອີ່ມີການປັບປຸງແພັ່ນດັ່ງກ່າວສາມາດອັນໄດ້

**os.W\_OK** ອີ່ມີການປັບປຸງແພັ່ນດັ່ງກ່າວສາມາດເຂີນໄດ້

`os.X_OK` คือ ทดสอบว่าแฟ้มดังกล่าวสามารถประมวลผลได้

ค่าที่ส่งกลับ: เป็นจริง เมื่อเป็นไปตามเงื่อนไขที่กำหนดไว้ในโหมด

ตัวอย่าง:

```
import os, sys, stat
Assuming C:\Python34\INPUT.txt exists, Set a file
execute by the group.
os.chmod("C:\Python34\INPUT.txt", stat.S_IXGRP)
Set a file write by others.
os.chmod("C:\Python34\INPUT.txt", stat.S_IWOTH)
print ("Changed mode successfully!!")
```

เอกสาร์พุต:



Changed mode successfully!!

OUTPUT

- **os.chdir()**: เปลี่ยนตำแหน่งที่อยู่ไปยังไดเรกทรอรี่ใหม่

รูปแบบคำสั่ง: `os.chdir(path)`

พารามิเตอร์: `path` คือ ที่อยู่ใหม่ของไดเรกทรอรี่ที่ต้องการย้ายไป

ค่าที่ส่งกลับ: "ไม่มี"

ตัวอย่าง:

```
import os
path = "c:\windows"
Check current working directory.
cur_dir = os.getcwd()
print ("Current working directory %s" % cur_dir)
Now change the directory
os.chdir(path)
Check current working directory.
cur_dir = os.getcwd()
print ("Directory changed successfully %s" % cur_dir)
```

เอกสาร์พุต:



Current working directory C:\Python34  
Directory changed successfully c:\windows

OUTPUT

- **os.chmod()**: เปลี่ยนสิทธิ์ของแฟ้มหรือไดเรกทรอรี่ (ใช้บัญนิกส์-ลิกกซ์)

รูปแบบคำสั่ง: `os.chmod(path, mod)`

พารามิเตอร์: `path` คือ ที่อยู่ของแฟ้ม หรือไดเรกทรอรี่ที่ต้องการเปลี่ยนสิทธิ์

`mode` คือ สิทธิ์ที่ต้องการกำหนด ดังนี้

`stat.S_ISUID` กำหนดค่า `UID` (User ID) ให้แฟ้มข้อมูล

stat.S\_ISGID กำหนดค่า GID (Group ID) ให้แฟ้มข้อมูล  
stat.S\_ENFMT สั่งให้บันทึกการใช้งานแฟ้ม  
stat.S\_ISVTX บันทึกภาพข้อความหลังจากประมวลผล  
stat.S\_IREAD เจ้าของแฟ้มสามารถอ่านได้  
stat.S\_IWRITE เจ้าของแฟ้มสามารถเขียนได้  
stat.S\_IEXEC เจ้าของแฟ้มสามารถประมวลผลได้  
stat.S\_IRWXU เจ้าของแฟ้ม อ่าน เขียน และประมวลผลเพิ่มได้  
stat.S\_IRUSR เจ้าของแฟ้มสามารถอ่านได้  
stat.S\_IWUSR เจ้าของแฟ้มสามารถเขียนได้  
stat.S\_IXUSR เจ้าของแฟ้มสามารถประมวลผลได้  
stat.S\_IRWXG สมาชิกกลุ่มเดียวกับเจ้าของแฟ้ม อ่าน เขียน และประมวลผลเพิ่มได้  
stat.S\_IRGRP สมาชิกในกลุ่มสามารถอ่านได้  
stat.S\_IWGRP สมาชิกในกลุ่มสามารถเขียนได้  
stat.S\_IXGRP สมาชิกในกลุ่มสามารถประมวลผลได้  
stat.S\_IRWXO สมาชิกกลุ่มอื่นๆ สามารถอ่าน เขียน และประมวลผล  
stat.S\_IROTH สมาชิกกลุ่มอื่นๆ สามารถอ่านได้  
stat.S\_IWOTH สมาชิกกลุ่มอื่นๆ สามารถเขียนได้  
stat.S\_IXOTH สมาชิกกลุ่มอื่นๆ สามารถประมวลผลได้

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
#!/usr/bin/python
import os, sys, stat
Assuming /tmp/test.txt exists, Set a file execute
by the group.
os.chmod("/tmp/test.txt", stat.S_IXGRP)
Set a file write by others.
os.chmod("/tmp/test.txt", stat.S_IWOTH)
print "Changed mode successfully!!"
```

ผลลัพธ์:

 OUTPUT Changed mode successfully!!

- **os.chown():** เปลี่ยนสิทธิ์ผู้ครอบครองแฟ้มหรือไดเรกทรอรี (ใช้บนยูนิเก็ส์-ลินุกซ์)

รูปแบบคำสั่ง: os.chown(path, uid, gid)

พารามิเตอร์: path คือ แฟ้ม หรือไดเรกทอรี่ที่ต้องการเปลี่ยนสิทธิ์ผู้ครอบครอง  
 uid คือ หมายเลขที่ใช้ระบุตัวตนของเจ้าของแฟ้มหรือไดเรกทอรี่  
 gid คือ หมายเลขที่ใช้ระบุกลุ่มของเจ้าของแฟ้มหรือไดเรกทอรี่  
 ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
import os, sys
Assuming /tmp/test.txt exists.
To set owner uid = 99, gid = -1 (for group of
super user).
os.chown("/tmp/test.txt", 99, -1)
print "Changed ownership successfully!!"
```

เอาต์พุต:



Changed ownership successfully!!

OUTPUT

- **os.chroot()**: เปลี่ยนเดี๋ยวโลท์ (default) ไดเรกทอรีของ root (ใช้บนยูนิกซ์-ลินุกซ์)

รูปแบบคำสั่ง: os.chroot(path)

พารามิเตอร์: path คือ ไดเรกทอรีใหม่ที่ต้องการเปลี่ยนเป็นเดี๋ยวโลท์ไดเรกทอรี

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
#!/usr/bin/python
import os, sys
To set the current root path to /tmp/usr
os.chroot("/tmp/usr")
print "Changed root path successfully!!"
```

เอาต์พุต:



Changed root path successfully!!

OUTPUT

- **os.getcwd()**: แสดงไดเรกทอรีที่ทำงานอยู่ในปัจจุบัน

รูปแบบคำสั่ง: os.getcwd()

พารามิเตอร์: ไม่มี

ค่าที่ส่งกลับ: ไดเรกทอรีปัจจุบัน

ตัวอย่าง:

```
import os, sys
Print current working directory
print ("Current working dir : %s" % os.getcwd())
```

ເອົາຕີ່ພຸດ:



OUTPUT

Current working dir : C:\Python34

- **os.link():** ສ້າງ shortcut ຂອງແພີມຂໍ້ມູນລົບ

ຮູບແບບຄຳສັ່ງ: `os.link(src, dst)`

ພາຣາມີເຕືອຮ່ວ: `src` ດີວ່າ ແພີມຕົ້ນຈົບທີ່ຕ້ອງການສ້າງ shortcut

`dst` ດີວ່າ shortcut ໄໝທີ່ເຊື່ອມໂຢງໄປຢັງແພີມ `src`

ຄ່າທີ່ສັງກັບ: `ໄມ່ມີ`

ຕົວຢ່າງ:

```
import os, sys
Open a file
src = "C:\Python34\INPUT.txt"
fd = os.open(src, os.O_RDWR|os.O_CREAT)
Close opened file
os.close(fd)
Now create another copy of the above file.
dst = "C:\Python34\LINK_INPUT.txt"
os.link(src, dst)
print ("Created hard link successfully!!")
```

ເອົາຕີ່ພຸດ:



OUTPUT

Created hard link successfully!!

- **os.listdir():** ແສດງຮັບຊື່ແພີມແລະໄດ້ເຮັດວຽກໃນໄດ້ເຮັດວຽກທີ່ກຳລັງທຳງານອູ້ງໂດຍໄມ່ຮັມໄດ້ເຮັດວຽກຮີ້ໜິດ . ແລະ ..

ຮູບແບບຄຳສັ່ງ: `os.listdir(path)`

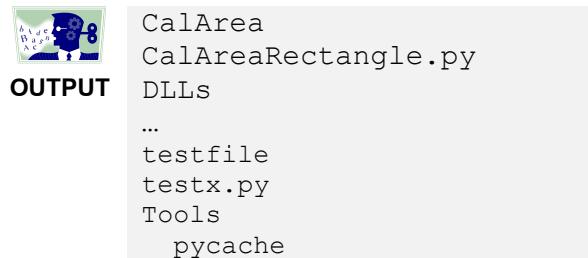
ພາຣາມີເຕືອຮ່ວ: `path` ດີວ່າ ຊື່ໄດ້ເຮັດວຽກທີ່ຕ້ອງການແສດງຮັບຊື່ແພີມ ແລະໄດ້ເຮັດວຽກຮີ້ໝາຍ

ຄ່າທີ່ສັງກັບ: ຮາຍການແພີມ ແລະໄດ້ເຮັດວຽກຮີ້ໝາຍທັງໝົດ

ຕົວຢ່າງ:

```
import os, sys
path = "C:\Python34"
dirs = os.listdir(path)
This would print all the files and directories
for file in dirs:
 print (file)
```

ເອົາຕີ່ພຸດ:



- **os.makedirs()**, **os.makedirs()** : สร้างไดเรกทอรีใหม่

รูปแบบคำสั่ง: `os.makedirs(path [,mode])`, `os.makedirs(path [,mode])`

พารามิเตอร์: `path` คือ ชื่อไดเรกทอรีที่ต้องการสร้างใหม่

`mode (option)` เป็นการกำหนดสิทธิ์ให้กับไดเรกทอรีที่สร้างขึ้นใหม่

ค่าที่ส่งกลับ: "ไม่มี"

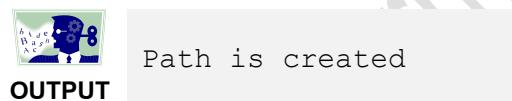
ตัวอย่าง:

```

import os, sys
Path to be created
path = "C:\Python34\TEST"
os.makedirs(path, 755);
print ("Path is created")

```

ผลลัพธ์:



- **os.open()**: เปิดแฟ้มข้อมูลเพื่ออ่าน-เขียน (บันยูนิกซ์-ลินก์)

รูปแบบคำสั่ง: `os.open(file, flags[, mode])`

พารามิเตอร์: `file` คือ ชื่อแฟ้มที่ต้องการอ่าน-เขียน

`flags` คือ โหมดในการเปิดแฟ้ม (เป็นค่าคงที่) ดังนี้

`os.O_RDONLY` เปิดเพื่ออ่านอย่างเดียว

`os.O_WRONLY` เปิดเพื่อเขียนเท่านั้น

`os.O_RDWR` เปิดเพื่ออ่าน และเขียน

`os.O_NONBLOCK` ไม่ปิดกันแฟ้มที่ต้องการเปิดใช้งาน

`os.O_APPEND` เปิดเพื่อเขียนข้อมูลเพิ่ม

`os.O_CREAT` เปิดแฟ้ม ถ้าไม่มีแฟ้มให้สร้างใหม่

`os.O_TRUNC` ตัด หรือลดขนาดแฟ้มให้มีขนาดเท่ากับ 0

`os.O_EXCL` สร้างแฟ้มใหม่ ถ้ามีแฟ้มอยู่แล้ว จะแสดงข้อผิดพลาด

`os.O_SHLOCK` ไม่อนุญาตให้แชร์แฟ้มข้อมูล

`os.O_EXLOCK` เปิดแฟ้มชนิด exclusive lock

`os.O_DIRECT` ไม่ส่งใจผลกรอบที่จาก cache ขณะเปิดแฟ้ม

`os.O_FSYNC` เขียนแฟ้มชนิดเข้าจังหวะ (synchronous)

`os.O_NOFOLLOW` ไม่สนใจการนำ `symlinks`

`mode (option)` ทำงานเหมือนกับ `mode` ในคำสั่ง `chmod`

ค่าที่ส่งกลับ: อ้อปเจกที่อ้างอิงไปยังแฟ้มข้อมูลจริง (File descriptor : แสดงรายละเอียดโครงสร้างแฟ้ม)

ตัวอย่าง:

```
import os, sys
Open a file
fd = os.open("INPUT.txt", os.O_RDWR|os.O_CREAT)
Write one string
os.write(fd, "This is test")
Close opened file
os.close(fd)
print ("Closed the file successfully!!")
```

เอกสาร์พุต:



OUTPUT

Closed the file successfully!!

- `os.read()` : อ่านข้อมูลจากแฟ้มทำงานร่วมกับ File descriptor (บัญชีนิกส์-ลินิกซ์)

รูปแบบคำสั่ง: `os.read(fd, n)`

พารามิเตอร์: `fd` คือ File descriptor (แสดงรายละเอียดโครงสร้างแฟ้ม)

`n` จำนวนไบต์ข้อที่ต้องการอ่าน

ค่าที่ส่งกลับ: สายอักขระที่เรียงต่อกัน (ข้อมูลเป็นไบต์เรียงต่อกัน)

ตัวอย่าง:

```
import os, sys
Open a file
fd = os.open("INPUT.txt",os.O_RDWR)
Reading text 12 bytes
ret = os.read(fd, 12)
print (ret)
Close opened file
os.close(fd)
print ("Closed the file successfully!!")
```

เอกสาร์พุต:



OUTPUT

b'first line\n'  
Closed the file successfully!!

- **os.remove()** : ลบแฟ้มข้อมูล

รูปแบบคำสั่ง: `os.remove(path)`

พารามิเตอร์: `path` คือ ชื่อแฟ้มที่ต้องการลบจากระบบ

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
import os, sys
listing directories
print ("The dir is: %s" %os.listdir(os.getcwd()))
removing
os.remove("t.txt")
listing directories after removing path
print ("The dir after removal of path : %s"
%os.listdir(os.getcwd()))
```

ผลลัพธ์:



**OUTPUT**

```
The dir is: ['CalArea', 'CalAreaRectangle.py',
'DLLs', 'Doc', 'exam10_1.py', 'exam10_2.py',
'exam10_9.py', t.txt,...]
The dir is: ['CalArea', 'CalAreaRectangle.py',
'DLLs', 'Doc', 'exam10_1.py', 'exam10_2.py',
'exam10_9.py',...]
```

- **os.removedirs()** : ลบไดเรกทรอรี

รูปแบบคำสั่ง: `os.removedirs(path)`

พารามิเตอร์: `path` คือ ชื่อดirektoryที่ต้องการลบจากระบบ

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
import os, sys
listing directories
print ("The dir is: %s" %os.listdir(os.getcwd()))
removing
os.removedirs("C:\Python34\TEST")
listing directories after removing directory
print ("The dir after removal is:"
%os.listdir(os.getcwd()))
```

ผลลัพธ์:



**OUTPUT**

```
The dir is: ['CalArea', 'CalAreaRectangle.py',
'DLLs', 'Doc', 'exam10_1.py', TEST,...]
...
The dir after removal is: ['CalArea',
'CalAreaRectangle.py', 'DLLs', 'Doc',
'exam10_1.py',...]
```

- **os.rename()** : เปลี่ยนชื่อแฟ้มหรือไดเรกทรอรี่

รูปแบบคำสั่ง: `os.rename(src, dst)`

พารามิเตอร์: `src` คือ ชื่อแฟ้ม หรือไดเรกทรอรี่เดิมที่ต้องการเปลี่ยนชื่อ  
`dst` คือ ชื่อแฟ้ม หรือไดเรกทรอรี่ใหม่

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
import os, sys
listing directories
print ("The dir is: %s"%os.listdir(os.getcwd()))
renaming directory
os.rename("TEST", "TEST1")
print ("Successfully renamed.")
listing directories after renaming
print ("The dir after rename is: %s"
%os.listdir(os.getcwd()))
```

ผลลัพธ์:

|                                                                                                     |                                                                                                                                                                                                                            |
|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <br><b>OUTPUT</b> | <pre>The dir is: ['CalArea', 'CalAreaRectangle.py', 'DLLs', 'Doc', 'exam10_1.py', <b>TEST</b>, ...] ... The dir after rename is: ['CalArea', 'CalAreaRectangle.py', 'DLLs', 'Doc', 'exam10_1.py', <b>TEST1</b>, ...]</pre> |
|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- **os.renames()** : เปลี่ยนชื่อแฟ้มหรือไดเรกทรอรี่เหมือน `rename` แต่แตกต่างที่ `renames` สามารถเปลี่ยนชื่อพร้อมกับย้ายไดเรกทรอรี่ใหม่ด้วย

รูปแบบคำสั่ง: `os.rename(old, new)`

พารามิเตอร์: `old` คือ ชื่อแฟ้ม หรือไดเรกทรอรี่เดิมที่ต้องการเปลี่ยนชื่อ  
`new` คือ ชื่อแฟ้ม หรือไดเรกทรอรี่ใหม่

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
import os, sys
print ("Current directory is: %s" %os.getcwd())
listing directories
print ("The dir is: %s"%os.listdir(os.getcwd()))
renaming file "t.txt"
os.renames("t.txt", "newdir/tt.txt")
print ("Successfully renamed.")
listing directories after renaming and moving
"t.txt"
print ("The dir is: %s" %os.listdir(os.getcwd()))
```

ผลลัพธ์:


**OUTPUT**

```
Current directory is: C:\Python34
The dir is: ['CalArea',
'CalAreaRectangle.py',...,t.txt,...]
Successfully renamed.
The dir is: ['CalArea',
'CalAreaRectangle.py',...,t.txt,...,newdir,...]
```

- **os.rmdir()** : ลบไดเรกทอรี่ (ต้องเป็นไดเรกทอรีว่าง)

รูปแบบคำสั่ง: `os.rmdir(path)`

พารามิเตอร์: `path` คือ ชื่อแฟ้ม หรือไดเรกทอรีเดิมที่ต้องการลบ

ค่าที่ส่งกลับ: "ไม่มี"

ตัวอย่าง:

```
import os, sys
listing directories
print "the dir is: %s" %os.listdir(os.getcwd())
removing path
os.rmdir("mydir")
listing directories after removing directory path
print "the dir is:" %os.listdir(os.getcwd())
```

เอกสารพูด:


**OUTPUT**

```
The dir is: ['CalArea',
'CalAreaRectangle.py',...,t.txt,...,newdir,...]
The dir is: ['CalArea',
'CalAreaRectangle.py',...,t.txt,...]
```

- **os.stat()** : แสดงสถิติแฟ้มข้อมูลจากการเรียกใช้งานของระบบปฏิบัติ

รูปแบบคำสั่ง: `os.stat(path)`

พารามิเตอร์: `path` คือ ชื่อแฟ้มที่ต้องการแสดงสถิติ

ค่าที่ส่งกลับ: รายการข้อมูลสถิติ

`st_mode` บิตข้อมูลที่ใช้สำหรับกำหนดความปลอดภัย

`st_ino` หมายเลข inode

`st_dev` ชื่ออุปกรณ์

`st_nlink` จำนวนลิงค์ที่เชื่อมต่อ

`st_uid` UID (user ID) ของผู้เป็นเจ้าของแฟ้ม

`st_gid` GID (group ID) ของผู้เป็นเจ้าของแฟ้ม

`st_size` ขนาดของแฟ้ม (มีหน่วยเป็นไบต์)

`st_atime` วัน-เวลาที่เข้าใช้งานล่าสุด

`st_mtime` วัน-เวลาที่ปรับปรุงแฟ้มล่าสุด

`st_ctime` วัน-เวลาที่ปรับปรุงแฟ้ม metadata ล่าสุด

ตัวอย่าง:

```
import os, sys
showing stat information of file "testx.py"
statinfo = os.stat('testx.py')
print (statinfo)
```

ເອກົດ:



**OUTPUT**

```
os.stat_result(st_mode=33206, st_ino=1407374883790118,
st_dev=1222005554, st_nlink=1, st_uid=0, st_gid=0,
st_size=113, st_atime=1388819475, st_mtime=1389107870,
st_ctime=1388819475)
```

- **os.utime()** : ກໍາທັນດວນ-ເວລາ ໃນການເຂົ້າໃຈໜານແລະປ່ຽນປ່ຽນແພັນ

ຮູບແບບຄຳສັ່ງ: `os.utime(path[, times])`

ພາຣາມີເຕືອນ: `path` ຄື່ອ ຂຶ້ນແພັນທີ່ຕ້ອງການກໍາທັນດວນເວລາ

`time` ຄື່ອ ເວລາໃໝ່ທີ່ຕ້ອງການເປີ່ຍິນແປ່ງໄກ້ກັບແພັນ ເນື່ອໄວ່

ກໍາທັນດວນເວລາທີ່ປ່ຽນປ່ຽນໃໝ່ ຈະໃຫ້ເວລາບັງຈຸບັນ ປະກອບດ້ວຍ (`atime`,

`mtime`) ເຊັ່ນ (`accesstime` ແລະ `modifiedtime`) ມີຮູບແບບຄື່ອ

`ymmd HHMM`

ຄ່າທີ່ສັງກັນ: ໄນມື້

ຕົວຢ່າງ:

```
import os, sys
Showing stat information of file
stinfo = os.stat('testx.py')
print (stinfo)
Using os.stat to receive atime and mtime of file
print ("access time of testx.py: %s"
%stinfo.st_atime)
print ("modified time of testx.py: %s"
%stinfo.st_mtime)
Modifying atime and mtime
os.utime("testx.py", (1330712280, 1330712292))
print ("done!!")
```

ເອກົດ:



**OUTPUT**

```
os.stat_result(st_mode=33206, st_ino=1407374883790118,
st_dev=1222005554, st_nlink=1, st_uid=0, st_gid=0,
st_size=358, st_atime=1388819475, st_mtime=1389109069,
st_ctime=1388819475)
access time of testx.py: 1388819475.676249
modified time of testx.py: 1389109069.4353268
```

- **os.walk()** : ແສດງໂຄຮງສ້າງຂອງໄດເຮັດວອຣີໃນຮູບແບບຂອງທີ່

ຮູບແບບຄຳສັ່ງ: `os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])`

ພາຣາມີເຕືອນ: `top` ກໍາທັນໄດເຮັດວອຣີຮາກ (root) ທີ່ຕ້ອງການແສດງຜລ (ຮາຍການທີ່ໄດ້ຮັບກລັບມາຈະເກີບໃນທັພເພີລ ມີ 3 ຊົນິດ ຄື່ອ `dirpath`, `dirnames`, `filenames`)

Topdown (option) กำหนดให้เป็น True เมื่อต้องการแสดงทรีจากบน

ลงล่าง และ False แสดงจากล่างขึ้นบน

Onerror แสดงความผิดพลาด ขณะแสดงไดเรคทรอรี่

Followlinks เมื่อกำหนดเป็น True จะแสดง symlinks

ค่าที่ส่งกลับ: ไม่มี

ตัวอย่าง:

```
import os
for root, dirs, files in os.walk(".",
topdown=False):
 for name in files:
 print(os.path.join(root, name))
 for name in dirs:
 print(os.path.join(root, name))
```

เอกสารพูด:



OUTPUT

```
.\CalArea__pycache__\CalAreaRectangle.cpython-34.pyc
.\CalArea__pycache__\CalAreaTriangle.cpython-34.pyc
...
```

- **os.write()**: เขียนข้อมูลลงแฟ้ม

รูปแบบคำสั่ง: `os.write(fd, str)`

พารามิเตอร์: `fd` คือ File descriptor สำหรับใช้อ้างอิงไปยังแฟ้มจริง  
`str` строкหรือสายอักขระที่ต้องการเขียนลงแฟ้ม

ค่าที่ส่งกลับ: จำนวนข้อมูลที่ได้เขียนลงแฟ้มแล้ว (มีหน่วยเป็นไบต์)

ตัวอย่าง:

```
import os, sys
Open file
fd = os.open("TEST.txt", os.O_RDWR|os.O_CREAT)
Writing text
ret = os.write(fd, 'This is test')
ret consists of number of bytes written to
TEST.txt
print ("the number of bytes written: ")
print (ret)
print ("written successfully")
Close opened file
os.close(fd)
print ("Closed the file successfully!!")
```

เอกสารพูด:



OUTPUT

```
the number of bytes written:
```

```
12
```

```
written successfully
```

```
Closed the file successfully!!
```

## បរទេសការណ៍

1. Alchin M, Pro Python, Apress, June 2010.
2. Allen Downey, J. E. a. C. M., Think Python: How to Think Like a Computer Scientist. Needham, Massachusetts, Green Tea Press, June 2014.
3. Allen, S., "Python." Retrieved April 12, 2014, from <http://www.dotnetperls.com/python>.
4. applicatio d., "An Introduction To Tkinter." Retrieved June 19, 2014, from <http://effbot.org/tkinterbook/>.
5. B.P, A. K., Learning Maths & Science using Python and writing them in LATEX. New Delhi 110067, Inter University Accelerator Centre, June 2010.
6. Beazley, D. M., Python Essential Reference, 4th Edition, Addison-Wesley Professional, July 2009.
7. Central, P., "Python Programming Guides and Tutorials." Retrieved May 22, 2014, from <http://www.pythoncentral.io/>.
8. Chun, W., Core Python Programming, 2nd Edition, Prentice Hall, September 2006.
9. Course, P., "Python3 Tutorial." Retrieved 15 February, 2014, from <http://www.python-course.eu/>.
10. Curtin, B., "Python 3 Porting Guide." Retrieved May 15, 2014, from [http://docs.pythonsprints.com/python3\\_porting/py-porting.html](http://docs.pythonsprints.com/python3_porting/py-porting.html).
11. Floyd, P., Dive Into Python 3, 2nd edition, Apress, November 2009.
12. Foundation, P. S., " Python 3.4.2 Documentation." Retrieved December 2013, 2014, from <https://docs.python.org/3/contents.html>.
13. Foundation, P. S., "Python v3.1.5 documentation." Retrieved January 2, 2014, from <https://docs.python.org/3.1/index.html>.
14. Foundation, P. S., "Python Documentation by Version." Retrieved January 6, 2014, from <https://www.python.org/doc/versions/>.
15. Grayson, J. E., Python and Tkinter Programming. Lafayette Place, Greenwich, the United States of America, Manning Publications Co., 2000.
16. Grayson, J. E., Python and Tkinter Programming. the United States of America, Manning Publications, January 2000.

17. Halterman, R. L., LEARNING TO PROGRAM WITH PYTHON, Southern Adventist University, 2011.
18. Hetland, M. L., Beginning Python: From Novice to Professional, 2nd edition, Apress, 2005
19. Hong, K. (2014). "PYTHON HOME 2014." Retrieved June 15, 2014, from <http://www.bogotobogo.com/python/pytut.php>.
20. java2s.com, "Python examples." Retrieved March 18, 2014, from <http://www.java2s.com/Code/Python/CatalogPython.htm>.
21. John Goerzen , B. R., Foundations of Python Network Programming, 2nd edition, Apress, December 2010.
22. Katja Schuerer, C. M., Catherine Letondal, Eric Deveaud, Introduction to Programming using Python, Pasteur Institute, February 2008.
23. Kiusalaas, J., Numerical Methods in Engineering with Python, 2nd Edition, Cambridge University Press, August 2014.
24. Lamber, K. A., Fundamentals of Python: From First Programs Through Data Structures, Cengage Learning, March 2011.
25. Lott, S. F., Building Skills in Python, April 2010.
26. Lundh, F., "An Introduction to Tkinter." Retrieved May 5, 2014, from <http://www.scoberlin.de/content/media/http/informatik/tkinter/>.
27. Matloff, N., Introduction to Network Programming with Python, University of California, Davis, May 2009.
28. Payne, J., Beginning Python: Using Python 2.6 and Python 3.1, February 2010.
29. Phillips, D., Python 3 Object Oriented Programming, Packt Publishing, July 2010.
30. Pimpler, E., Programming ArcGIS 10.1 with Python Cookbook, Packt Publishing ([www.packtpub.com](http://www.packtpub.com)), 2013.
31. Pimpler, E., Programming ArcGIS 10.1 with Python Cookbook, Packt Publishing, February 2013.
32. Project, T. G., "The Python GTK+ 3 Tutorial." Retrieved March, 2014, from <http://python-gtk-3-tutorial.readthedocs.org/en/latest/index.html>.
33. Roseman, M., "TkDocs." Retrieved May 28, 2014, from <http://www.tkdocs.com/tutorial/>.

34. Sentance, S., "Python School." Retrieved April 25, 2014, from  
<http://www.pythonschool.net/>.
35. Shaw, Z. A., "Learn Python the Hard Way." 3rd Edition. Retrieved May 20, 2014, from <http://learnpythonthehardway.org/book/index.html>.
36. Shipman, J. W., Tkinter 8.5 reference: a GUI for Python. New Mexico, New Mexico Institute of Mining and Technology, 2013.
37. Summerfield, M., Programming in Python 3: A Complete Introduction to the Python Language, 2nd edition, Addison-Wesley Professional, November 2009.
38. Summerfield, M., Rapid GUI Programming with Python and Qt, Prentice Hal, October 2007.
39. Thoreau, H. D., Introduction to Computers, Internet and World Wide Web, Deitel & Associates, Inc, December 2005.
40. Tim Hall , J.-P. S., Python 3 for Absolute Beginners, Apress, October 2009.
41. Tiponut, S. V., Python Network Programming. Romania, Technical University Timisoara, 2001.
42. Tutorialspoint, "Python Tutorial." Retrieved May 17, 2014, from  
<http://www.tutorialspoint.com/python/>.
43. Wikipedia, "History of Python." Retrieved November 12, 2013, from  
[http://en.wikipedia.org/wiki/History\\_of\\_Python](http://en.wikipedia.org/wiki/History_of_Python).
44. Zelle, J. M., Python Programming: An Introduction to Computer Science, Franklin Beedle & Associates, December 2003.

## ดัชนีคำศัพท์

### ก

|                                                                   |     |
|-------------------------------------------------------------------|-----|
| กระบวนการทำงาน (Documentation/Procedure)                          | 3   |
| การจัดการหน่วยความจำอัตโนมัติ (Garbage collection)                | 19  |
| การประกาศตัวแปร (Variable declaration)                            | 64  |
| กำหนดค่าให้ตัวแปร (Assigning values to variables)                 | 64  |
| การใช้ตัวแปร (Use the variable)                                   | 65  |
| การเปลี่ยนค่าตัวแปร (Forcing a number type)                       | 70  |
| การควบคุมทิศทางแบบเลือกทำ (Decisions, Choice, Selection)          | 114 |
| การวนซ้ำแบบไม่รู้จบ (The Infinite Loop)                           | 130 |
| การประกาศฟังชัน (Defining a function)                             | 156 |
| การเรียกใช้ฟังชัน (Calling a function)                            | 157 |
| การส่งผ่านอาร์กิวเมนต์ (Pass by reference vs value)               | 158 |
| การสร้างฟังก์ชันโดยไม่ระบุชื่อ (The Anonymous functions: lambda)  | 166 |
| การส่งค่ากลับจากฟังชัน (The return statement)                     | 171 |
| การส่งค่ากลับจากฟังชันหลายค่า (Returning multiple values)         | 175 |
| การเรียกตัวเอง หรือการเวียนเกิด (Recursion)                       | 178 |
| การจัดการข้อผิดพลาด (Exceptions handling)                         | 193 |
| การตรวจสอบความผิดพลาดแบบหลาย Excetions (Multiple exceptions)      | 197 |
| การสร้าง Exception ขึ้นมาใช้งานเอง (User-defined exceptions)      | 207 |
| การยืนยันในสมมติฐาน (Assertions)                                  | 208 |
| การเปิดแฟ้ม (Opening files)                                       | 214 |
| การปิดแฟ้ม (Opening files)                                        | 216 |
| การโปรแกรมเชิงวัตถุ (Object-Oriented Programming Concept: OOP)    | 231 |
| การห่อหุ้มข้อมูล/การซ่อนข้อมูล (Encapsulation/Data hinding)       | 237 |
| การสืบทอด (Inheritance)                                           | 237 |
| การพ้องรูป (Polymorphism)                                         | 238 |
| การสร้างคลาส (Creating class)                                     | 239 |
| การสร้างอินสแตนซ์ของวัตถุ (Creating instance objects)             | 246 |
| การเข้าถึงแอ็ตทริบิวต์และเมธอด (Accessing attributes and methods) | 248 |
| การลบวัตถุที่สร้างขึ้น (Destroying objects)                       | 251 |
| การสืบทอดคุณสมบัติของคลาส (Class Inheritance)                     | 253 |
| การโverbOverride ดึงเมธอด (Overriding methods)                    | 260 |

|                                                             |     |
|-------------------------------------------------------------|-----|
| การโอเวอร์โหลดดึงเมธอด (Overloading method/function)        | 263 |
| การโอเวอร์โหลดตัวดำเนินการ (Overloading operators)          | 265 |
| การห่อหุ้มข้อมูล/การซ่อนข้อมูล (Encapsulation/Data hiding)  | 268 |
| การเพิ่มระเบียน (Insert)                                    | 313 |
| การลบระเบียน (Delete)                                       | 316 |
| การปรับปรุงระเบียน (Update)                                 | 315 |
| การอ่านระเบียน (Read)                                       | 314 |
| การยืนยันคำสั่ง (Commit)                                    | 317 |
| การยกเลิกการทำงานของคำสั่งก่อนหน้า (Rollback)               | 317 |
| การจัดเรียงข้อมูล (Sorting)                                 | 421 |
| การค้นหาข้อมูล (Searching)                                  | 431 |
| การพร็อตกราฟ (Plot Graph)                                   | 469 |
| การจัดการรูปทรงเลขากณิตให้กับ Widgets (Geometry management) | 483 |
| การจัดการกับเหตุการณ์ต่างๆ (Event Handling)                 | 483 |
| การตอบสนองต่อเหตุการณ์ที่เกิดขึ้น (Command Callbacks)       | 483 |
| การจัดวาง Widgets ด้วยเลขากณิต (Geometry management)        | 487 |
| เกมส์ Tic-Tac-Toe                                           | 559 |

## ข

|                                               |     |
|-----------------------------------------------|-----|
| ข้อมูล/สารสนเทศ (Data/Information)            | 3   |
| ข้อผิดพลาดทางไวยากรณ์ (Syntax error)          | 7   |
| ขั้นตอนการพัฒนาโปรแกรม (Software development) | 8   |
| ข้อมูลพื้นฐาน (Basic data types)              | 66  |
| ข้อมูลตัวเลข (Numeric)                        | 66  |
| ข้อมูลชนิดสายอักษร (String)                   | 70  |
| ข้อมูลเชิงประกอบ (Composite data types)       | 77  |
| ขอบเขตของตัวแปร (Scope of variables)          | 176 |
| ข้อผิดพลาด (Exceptions)                       | 192 |
| ข้อความ (Message)                             | 526 |

## ค

|                                                                          |    |
|--------------------------------------------------------------------------|----|
| คอมไพล์เลอร์ (Compiler)                                                  | 6  |
| โครงสร้างการทำงานแบบตามลำดับ (Sequence)                                  | 11 |
| โครงสร้างการทำงานแบบเลือกตัดสินใจ หรือเงื่อนไข (Decision หรือ Selection) | 11 |
| โครงสร้างการทำงานแบบการทำซ้ำ (Repetition หรือ Loop)                      | 13 |
| คำสั่งช่วยเหลือ help ()                                                  | 62 |

|                                                                   |     |
|-------------------------------------------------------------------|-----|
| ค่าคงที่ (Literal constants)                                      | 63  |
| คำส่วน (Reserved word, Keyword)                                   | 66  |
| ควบคุมทิศทางแบบวนรอบ หรือทำซ้ำ (Loop, Iteration)                  | 128 |
| คำสั่ง break                                                      | 138 |
| คำสั่ง continue                                                   | 140 |
| คำสั่ง pass                                                       | 141 |
| คอนสตรัคเตอร์ (Constructor)                                       | 239 |
| เคอร์เซอร์ (Cursors)                                              | 496 |
| เครื่องคิดเลขขนาดเล็ก (Mini-Calculator)                           | 554 |
| <b>จ</b>                                                          |     |
| จำนวนตรรกะ (Boolean)                                              | 69  |
| จำนวนเชิงซ้อน (Complex numbers)                                   | 69  |
| <b>ช</b>                                                          |     |
| ชนิดข้อมูล (Data types)                                           | 66  |
| <b>ซ</b>                                                          |     |
| ซอฟต์แวร์ระบบ (System software)                                   | 2   |
| ซอฟต์แวร์ที่ควบคุมอุปกรณ์ฮาร์ดแวร์โดยตรง (Device driver program)  | 2   |
| ซอฟต์แวร์ประยุกต์ (Application software)                          | 3   |
| ซอฟต์แวร์เสรี (Open source software)                              | 19  |
| เซต (Sets)                                                        | 90  |
| ซิงโครไนซ์เทรด (Threads Synchronization)                          | 351 |
| <b>ธ</b>                                                          |     |
| ฐานข้อมูล (Database)                                              | 306 |
| <b>ด</b>                                                          |     |
| ติกซันนารี (Dictionary)                                           | 86  |
| ดีคอนสตรัคเตอร์ (Deconstructor)                                   | 239 |
| <b>ต</b>                                                          |     |
| ตัวแปร (Variable)                                                 | 63  |
| ตัวเลขทศนิยม หรือจำนวนจริง (Floating-Point numbers)               | 68  |
| ตัวดำเนินการพิเศษเกี่ยวกับสตริง (String special operators)        | 76  |
| ตัวดำเนินการพื้นฐานของลิสต์ (Basic list operations)               | 79  |
| ตัวแปรชนิดเปลี่ยนแปลงข้อมูลได้ตลอดเวลา (Mutable)                  | 86  |
| ตัวดำเนินการที่ใช้เปรียบเทียบสำหรับเซต (Set comparison operators) | 94  |

|                                                          |     |
|----------------------------------------------------------|-----|
| ตัวดำเนินการ (Operator)                                  | 100 |
| ตัวถูกดำเนินการ (Operand)                                | 100 |
| ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)         | 100 |
| ตัวดำเนินการทางด้านการเปรียบเทียบ (Comparison Operators) | 102 |
| ตัวดำเนินการกำหนดค่า (Assignment Operators)              | 103 |
| ตัวดำเนินการระดับบิต (Bitwise Operators)                 | 105 |
| ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)            | 108 |
| ตัวดำเนินงานการเป็นสมาชิก (Membership Operators)         | 109 |
| ตัวดำเนินการเอกลักษณ์ (Identity Operators)               | 110 |
| <b>ก</b>                                                 |     |
| ทัพเพิล (Tuples)                                         | 81  |
| <b>ธ</b>                                                 |     |
| -thread และ -process (Threads and Processes)             | 345 |
| -thread (Thread)                                         | 346 |
| <b>น</b>                                                 |     |
| นิพจน์ (Expression)                                      | 100 |
| แนวคิดเชิงวัตถุ (Object)                                 | 232 |
| นิพจน์ปกติ (Regular Expression: Regex)                   | 272 |
| นำเข้าข้อมูล (Entry)                                     | 512 |
| <b>บ</b>                                                 |     |
| บุคคลากร (Peopleware)                                    | 3   |
| บิตแมป (Bitmaps)                                         | 495 |
| <b>ป</b>                                                 |     |
| โปรแกรมคอมพิวเตอร์ (Computer programs)                   | 2   |
| ปุ่ม (Button)                                            | 501 |
| <b>ผ</b>                                                 |     |
| ผังงาน (Flowchart)                                       | 9   |
| ผังงานระบบ (System flowchart)                            | 9   |
| ผังงานโปรแกรม (Program flowchart)                        | 9   |
| ผลต่าง (Difference)                                      | 92  |
| ผลต่างสมมาตร (Symmetric difference)                      | 92  |
| ผู้ห่อหุ้ม (Wrapper)                                     | 457 |
| ผืนผ้าใบ (Canvas)                                        | 504 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| <b>พ</b>                                                     |     |
| แพ็คเกจ (Packages)                                           | 189 |
| Процесс (Процес)                                             | 345 |
| <b>ฟ</b>                                                     |     |
| ฟังก์ชัน (Function)                                          | 155 |
| แฟ้มข้อมูล (File)                                            | 213 |
| ฟิลด์ (Field)                                                | 306 |
| เฟรมข้อมูล (Data frame)                                      | 389 |
| ฟังก์ชันอัตราการเติบโต (Growth-rate functions)               | 415 |
| เฟรม (Frame)                                                 | 498 |
| <b>ก</b>                                                     |     |
| ภาษาคอมพิวเตอร์ (Computer languages)                         | 3   |
| ภาษาเครื่อง (Machine language)                               | 3   |
| ภาษาระดับต่ำ (Low level language)                            | 3   |
| ภาษาแอสแซมบลี (Assembly language)                            | 4   |
| ภาษาระดับสูง (High level language)                           | 4   |
| ภาษาระดับสูงมาก (Very high-level language)                   | 6   |
| ภาษาธรรมชาติ (Natural languages)                             | 7   |
| ภาษากราว (Glue language)                                     | 20  |
| ภาษาสคริปต์ (Scripting language)                             | 22  |
| <b>ม</b>                                                     |     |
| โมดูล (Module)                                               | 183 |
| เมธอด (Method)                                               | 239 |
| <b>ย</b>                                                     |     |
| ยูเนียน (Union)                                              | 92  |
| ยกเลิกการเชื่อมต่อ กับฐานข้อมูล (Disconnect)                 | 318 |
| <b>ร</b>                                                     |     |
| ระบบปฏิบัติการ (Operating system)                            | 2   |
| รหัสเทียม หรือซูโดโค้ด (Pseudo Code)                         | 14  |
| รหัสควบคุม (Escape Characters)                               | 75  |
| ระเบียน (Record)                                             | 306 |
| ระบบสารสนเทศภูมิศาสตร์ (Geographic Information System: GIS ) | 377 |
| <b>ล</b>                                                     |     |

|                                                                          |     |
|--------------------------------------------------------------------------|-----|
| เลขทศนิยม (Floating-point formatting)                                    | 59  |
| เลขจำนวนเต็ม (Integers)                                                  | 67  |
| ลำดับความสำคัญของตัวดำเนินการ (Operators Precedence)                     | 111 |
| ลูปซ้อน (Nested loops)                                                   | 152 |
| เลเยอร์ (Layers)                                                         | 390 |
| เลbel หรือป้ายชื่อ (Label)                                               | 516 |
| <b>ং</b>                                                                 |     |
| ไวยากรณ์ของภาษา (Syntax)                                                 | 6   |
| วัตถุ หรืออ็อกเจกต์ (Object)                                             | 51  |
| เว็บ (Web)                                                               | 319 |
| เวลาที่ใช้ในการประมวลผล (Processing Time)                                | 409 |
| <b>ঃ</b>                                                                 |     |
| สายอักขระ (String)                                                       | 56  |
| สัญลักษณ์ที่ใช้ในการแปลงสายอักขระ หรือสตริง (String formatting operator) | 57  |
| <b>ঁ</b>                                                                 |     |
| อุปกรณ์ฮาร์ดแวร์ (Hardware)                                              | 2   |
| แอสเซมเบลอร์ (Assembler)                                                 | 4   |
| օօպেজক্টপ্ৰগ্ৰাম (Object program)                                        | 7   |
| একক চীকাৎপ্ৰগ্ৰাম (Execute program)                                      | 7   |
| অন্তেরোৰ্প্ৰিটেৰ (Interpreter)                                           | 7   |
| অন্তেৱৰ্চেকচন (Intersection)                                             | 92  |
| অন্সট্ৰেণ্স (Instance)                                                   | 237 |
| এডত্ৰিবিত্ৰ/প্ৰোপেৰেটীস/দাতা (Attributes/Properties/Data)                | 239 |
| এডত্ৰিবিত্ৰ নিৰ্দেশ বিলোগ কোড (Built-in class attributes)                | 249 |
| ইপি এডেৰেস (IP Address)                                                  | 357 |
| অলগোৰিদম (Algorithm)                                                     | 409 |
| অত্রাগতিৰিক্ষণ (Big-O)                                                   | 418 |
| আৱেজ এবং মেথডগুৰু (Array and Matrix)                                     | 461 |

