อินเตอร์พริตเตอร์ไพธอนขนาดเล็กสำหรับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ

A Small Python Interpreter for The Programmable Browser on Mobile Phones

นายกฤษณ์ชัย วันชัยนาวิน
นายณัฐวุฒิ นาไวย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

อินเตอร์พริตเตอร์ไพธอนขนาดเล็กสำหรับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ

A Small Python Interpreter for The Programmable Browser on Mobile Phones

โดย

นายกฤษณ์ชัย วันชัยนาวิน

นายณัฐวุฒิ นาไวย์

อาจารย์ที่ปรึกษา

ดร. วิศิษฏ์ หิรัญกิตติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง อินเตอร์พริตเตอร์ไพธอนขนาดเล็กสำหรับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ

A Small Python Interpreter for The Programmable Browser on Mobile Phones

ผู้จัดทำ

1. นายกฤษณ์ชัย วันชัยนาวิน รหัสประจำตัว 43015350
2. นายณัฐวุฒิ นาไวย์ รหัสประจำตัว 43015357

_____ อาจารย์ที่ปรึกษา

(ดร. วิศิษฏ์ หิรัญกิตติ)

อินเตอร์พริตเตอร์ไพธอนขนาดเล็กสำหรับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ

นายกฤษณ์ชัย วันชัยนาวิน รหัส 43015350
นายณัฐวุฒิ นาไวย์ รหัส 43015357
ดร. วิศิษฏ์ หิรัญกิตติ อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

การใช้งานเว็บบราวเซอร์ในปัจจุบันยังเป็นลักษณะทำทีละขั้นควบคุมโดยผู้ใช้ คือบราวเซอร์จะต้องคอยรอรับ URL จากผู้ใช้เสมอ ก่อนที่เปลี่ยน URL นั้นเป็นคำร้องขอส่งไปยังเว็บเซิร์ฟเวอร์เพื่อเปิดหน้าเว็บ ในบางครั้งผู้ใช้ป้อน URL ให้โดยตรง แต่บางครั้งเป็นทางอ้อมโดยการติดตามลิงค์ จึงได้มีแนวคิดในการสร้างบราวเซอร์ที่สามารถทำงานได้เองแบบกึ่งอัตโนมัติ โดยบราวเซอร์นี้จะทำงานตามชุดคำสั่งที่โปรแกรมให้โดยผู้ใช้ในรูปของสคริปต์หรือชุดคำสั่ง โดยจะเรียกบราวเซอร์ชนิดนี้ว่า “บราวเซอร์ที่สามารถโปรแกรมได้”

เพื่อให้บราวเซอร์สามารถกระทำคำสั่งในสคริปต์ได้เราจำเป็นที่จะต้องมีการสร้างอินเตอร์พรีตเตอร์สำหรับประมวลผลคำสั่งนั้นๆ สำหรับปริญญานิพนธ์นี้เป็นการพัฒนาอินเตอร์พรีตเตอร์ให้กับบราวเซอร์ที่สามารถโปรแกรมได้สำหรับใช้บนเครื่องโทรศัพท์มือถือ โดยภาษาสคริปต์ที่เลือกใช้คือภาษาไพธอน เราจึงได้พัฒนาอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กขึ้นเพื่อการนี้
อินเตอร์พรีตเตอร์ที่พัฒนาในปริญญานิพนธ์นี้มีความสามารถในการแปล และกระทำชุดคำสั่งประโยคพื้นฐานภาษาไพธอน โดยมีองค์ประกอบของ ข้อมูลชนิดพื้นฐาน ตัวโอเปอเรเตอร์ ฟังก์ชันพื้นฐานในภาษา และประโยคเงื่อนไข ประโยคการทำซ้ำ แต่ไม่รวม การนิยามคลาสและฟังก์ชัน ทั้งนี้ได้มีการเพิ่มข้อมูลเกี่ยวกับเวลาเข้าไปในภาษาไพธอน สำหรับฟังก์ชันพื้นฐานในภาษาได้มีการเพิ่มฟังก์ชันการติดต่อสื่อสาร เช่น ฟังก์ชันการขอรับเอกสารผ่านทางเครือข่ายคอมพิวเตอร์โดยอาศัย URL ฟังก์ชันการส่งอีเมลล์ เป็นต้น

A Small Python Interpreter for The Programmable Browser on Mobile Phones

Kritchai Vanchainavin

Nuttawuth Nawai

Dr. Visit Hirankitti Advisor

Abstract

So far the way we use a conventional web browser has been rather a manual one. The user must take control over the browser by sending his/her demanded URL to the browser as the browser's request to the web server repeatedly. Sometimes we do that directly by posing an explicit URL, but sometimes indirectly by following a link, which is in fact an implicit URL. To design a capable browser, we prefer it to work semi-automatically, rather than manually, by following a sequence of instructions programmed as a script by its user. The browser performs a task by executing the script given by the user. We call this kind of browser, a "programmable browser".

For such a browser to be able to execute instructions from the script, it needs to employ a script interpreter. In this project we have developed an interpreter for a programmable browser on a mobile phone. The language of the script is based on Python. We, therefore, have developed a small Python interpreter to use in a programmable browser.

The interpreter is able to interpret basic Python statements containing most basic data types, operators, built-in functions, and control structures, but not classes and function definitions. We also extend the Python language with a new data type, i.e. a time data type. The built-in functions of our language are mainly communicating functions, e.g. functions for getting a document from the Net referred by a URL, functions for sending emails, and so on.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และความร่วมมือจากหลายๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ วิศิษฏ์ หิรัญกิตติ อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือทั้งทางด้านความรู้ และเครื่องมืออุปกรณ์ต่างๆ ที่จำเป็นต้องใช้ประกอบการทำปริญญานิพนธ์ฉบับนี้ ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้มีวันนี้ ซึ่งก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุกๆ ด้านอันหาที่เปรียบมิได้ คณะผู้จัดทำขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

กฤษณ์ชัย วันชัยนาวิน

ณัฐวุฒิ นาไวย์

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันนี้โทรศัพท์มือถือ นับได้ว่าเป็นอุปกรณ์สื่อสารที่ผู้คนส่วนใหญ่นิยมใช้กันอย่างแพร่หลาย ซึ่งไม่เพียงแต่การพูดคุยกันเท่านั้น แต่โทรศัพท์มือถือยังสามารถที่จะหาข้อมูลข่าวสารผ่านทางอินเตอร์เน็ตได้ด้วยซึ่งก็จะมีการทำงานผ่านบราวเซอร์ แต่โดยทั่วไปแล้วบราวเซอร์นั้นจะนำข้อมูลข่าวสารต่างๆ แต่ละครั้งมาได้นั้น จำเป็นที่จะต้องได้รับการร้องขอจากผู้ใช้โดยตรงเสมอจึงทำให้ไม่สะดวกในการใช้งานเท่าที่ควร และถูกจำกัดโดยที่ผู้ใช้ต้องมีการตอบโต้กับโปรแกรมเท่านั้นจึงจะสามารถใช้งานได้

จึงได้มีแนวคิดในการสร้างบราวเซอร์ที่สามารถโปรแกรมได้บนเครื่องโทรศัพท์มือถือขึ้น ซึ่งจะทำให้ผู้ใช้สามารถที่จะสั่งการต่างๆ ไว้ล่วงหน้าเพื่อให้บราวเซอร์สามารถทำงานตามรูปแบบที่ใช้กำหนดขึ้นมาได้ แต่ในการที่จะทำให้ทำให้บราวเซอร์นั้นสามารถโปรแกรมได้นั้นจำเป็นอย่างยิ่งที่จะต้องมีภาษาสคริปต์ตัวหนึ่งซึ่งทำหน้าที่เป็นเสมือนใบสั่งที่เตรียมเอาไว้ เพื่อให้ตัวบราวเซอร์นั้นสามารถทำงานได้ตามต้องการ โดยภาษาที่จะใช้ในการทำงานเช่นนี้จะต้องเป็นภาษาที่มีความยืดหยุ่น และเป็นภาษาสคริปต์ซึ่งทางผู้จัดทำได้เลือกรูปแบบของภาษาที่เหมาะกับการทำงานเช่นนี้ซึ่งก็คือภาษาไพธอน (Python) แต่เนื่องจากบนเครื่องโทรศัพท์มือถือนั้นไม่สามารถที่จะทำงานสคริปต์ของภาษาไพธอน ซึ่งหมายถึงไม่มีอินเตอร์พรีตเตอร์ที่จะมาจัดการกับสคริปต์ดังกล่าวจึงจำเป็นที่จะต้องสร้างอินเตอร์พรีตเตอร์ไพธอนสำหรับบราวเซอร์ที่สามารถโปรแกรมได้บนเครื่องโทรศัพท์มือถือขึ้นเพื่อที่จะจัดการกับสคริปต์ต่างๆ เพื่อสั่งให้บราวเซอร์ทำงานอีกที

โดยการสร้างตัวอินเตอร์พรีตเตอร์นี้จะเป็นการสร้างขึ้นเพื่อรองรับภาษาสคริปต์ซึ่งอ้างอิงจากรูปแบบของภาษาไพธอนเป็นส่วนใหญ่ และได้เพิ่มชนิดของข้อมูล และรูปแบบการทำงานบางอย่างเพิ่มเติมเข้าไปเพื่อให้เหมาะสมกับการใช้งานร่วมกับบราวเซอร์ ซึ่งจะทำให้บราวเซอร์บนมือถือที่สร้างขึ้นมาสามารถที่จะทำการเขียนสคริปต์ และรันสคริปต์ดังกล่าวได้ โดยสามารถเขียนคำสั่งพื้นฐานที่ทำงานแบบลำดับ (Sequential Statement) เช่นการกำหนดค่าให้กับตัวแปร (Assign Statement), คำสั่งการประมวลผลทางคณิตศาสตร์ อาทิ การบวก ลบ คูณ หาร เป็นต้น สามารถใช้คำสั่งการทำงานแบบวนรอบ (Loop Statement) เช่นคำสั่ง for while เป็นต้น สามารถใช้คำสั่งการทำงานแบบเงื่อนไข (Condition Statement) เช่น if else เป็นต้น และยังมีการสร้างฟังก์ชันพิเศษบางอย่างเพื่อให้สามารถทำงานร่วมกับบราวเซอร์ได้โดยตรงเพื่อใช้ในการเปิดยูอาร์แอลต่างๆ ที่ต้องการอีกด้วย

โดยการที่จะทำให้โปรแกรมมีความสามารถเข้าใจ หรือสามารถที่จะจัดการกับรูปแบบของคำสั่งต่างๆ ตามที่ได้กำหนดขึ้นมาได้นั้น จำเป็นที่จะต้องใช้ทฤษฎีการตรวจจับคำ และการตรวจสอบรูปแบบของประโยคคำสั่งต่างๆ ซึ่งมีลักษณะคล้ายคลึงกับการทำงานของคอมไพเลอร์ซึ่งจะมีการตรวจสอบว่าคำ

ต่างๆ นั้นถูกต้องตามรูปแบบที่กำหนดหรือไม่ และเมื่อนำคำต่างๆ มาประกอบกันเป็นประโยคแล้วนั้นถูกต้องตามรูปประโยคต่างๆ ที่ได้กำหนดไว้หรือไม่

และจากที่ได้กล่าวมาทั้งหมดนั้น ผู้อ่านคงจะได้เห็นถึงความไม่สะดวกสบายและการมีข้อจำกัดของบราวเซอร์แบบทั่วไป และประโยชน์ที่จะได้จากบราวเซอร์ที่สามารถโปรแกรมได้ซึ่งพัฒนาจากภาษาไพธอน แล้ว ซึ่งในรายละเอียดจะได้กล่าวต่อไป

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาหลักการทำงานของอินเตอร์พรีตเตอร์ และอินเตอร์พรีตเตอร์ภาษาไพธอน

1.2.2 เพื่อศึกาการเขียนและพัฒนาโปรแกรมจาวาบนโทรศัพท์มือถือ

1.2.3 เพื่อพัฒนาอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กเพื่อใช้บนโทรศัพท์มือ

1.2.4 เพื่อนำอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กนี้ไปใช้งานกับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ

1.3 ขอบเขตของงานวิจัย

1.3.1 สร้างอินเตอร์พรีตเตอร์ของภาษาที่อ้างอิงจากภาษาไพธอน โดยมีรูปแบบคำสั่งทั่วๆ ไปของภาษาไพธอน

1.3.2 อินเตอร์พรีตเตอร์สามารถอินเตอร์พรีตคำสั่งแบบลำดับ (sequential statement) เช่นคำสั่งการกำหนดค่า (assign statement), คำสั่งทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร เป็นต้น

1.3.3 อินเตอร์พรีตเตอร์สามารถอินเตอร์พรีตคำสั่งแบบวนรอบ (loop statement) เช่นคำสั่ง for while เป็นต้น

1.3.4 อินเตอร์พรีตเตอร์สามารถอินเตอร์พรีตคำสั่งแบบเงื่อนไข (condition statement) เช่นคำสั่ง if else เป็นต้น

1.3.5 อินเตอร์พรีตเตอร์สามารถอินเตอร์พรีตฟังก์ชั่นพื้นฐานต่างๆ ของภาษาไพธอนได้ เช่น print, range, len เป็นต้น

1.3.6 สร้างคำสั่งและชนิดของข้อมูลบางตัวขึ้นมาใหม่ขึ้นมาเพื่อให้สามารถทำงานร่วมกับบราวเซอร์ได้

1.4 วิธีการดำเนินงาน

การทำงานแบ่งออกเป็น 3 ส่วน ดังนี้

1.4.1 รวมรวม และศึกษาข้อมูลที่สำคัญต่างๆ ดังต่อไปนี้

1. ลักษณะและรูปแบบของคำสั่งในภาษาไพธอน
2. หลักการทำงานโดยรวมของอินเตอร์พรีตเตอร์
3. ทฤษฎีการตรวจจับคำ (lexical analysis)

4. ทฤษฎีการตรวจสอบประโยค (Syntax analysis)
5. การเขียนโปรแกรมจาวาบนโทรศัพท์มือถือ

1.4.2 วิเคราะห์ข้อมูลที่ได้ทำการรวบรวมมาทั้งหมด เพื่อกำหนดขอบเขตของโครงงาน

1.4.3 พัฒนาโครงงานจากขอบเขตที่กำหนดไว้ โดยมีขั้นตอนดังต่อไปนี้

1. ออกแบบโครงสร้างของโครงงาน
2. พัฒนาโปรแกรมขึ้นตามโครงสร้างที่ได้ออกแบบไว้
3. ทดสอบการทำงานคราวๆ ของโครงงานและทำการปรับปรุงแก้ไขโครงสร้างจนได้โครงสร้างที่ดีที่สุด
4. เพิ่มรายละเอียดเข้าไปยังจุดต่างๆ จนได้โปรแกรมที่สมบูรณ์แบบและครบถ้วน
5. ทำการทดสอบและแก้ไขจุดบกพร่องต่างๆ ของโปรแกรม

บทที่ 2
ภาษาไพธอน (Python)

ภาษาไพธอนคือภาษาสคริปต์ซึ่งมีการทำงานแบบอินเตอร์พรีตเตอร์, เป็นภาษาระดับสูง (High-level language), มีความสามารถเชิงวัตถุ (Object-oriented) และมีความสามารถในการเป็นภาษาสคริปต์ฝั่งเซิร์ฟเวอร์ (Server-side scripting language) มีการออกแบบให้ไม่ขึ้นอยู่กับระบบปฏิบัติการใดๆ จึงเหมาะกับการใช้งานเป็นภาษาสคริปต์ ภาษาไพธอนมีซินแทคที่ไม่สวยงามหรูหรา(Rich syntax)แต่เข้าใจได้ง่าย ซึ่งมีประโยชน์กว่าภาษาที่มีซินแทคที่สวยงามหรูหรา เหมือนดังที่ Guido van Rossum ผู้ให้กำเนิดภาษาไพธอนได้กล่าวไว้ว่า "ภาษาที่สวยงามหรูหราเป็นภาระมากกว่าที่จะช่วย" ภาษาไพธอนมีการพัฒนาอย่างต่อเนื่องโดยไพธอนออแกนไนเซชั่น(Python Organization)เวอร์ชั่นล่าสุดคือ 2.2.1 โดยมีผู้พัฒนาหลายแห่งให้การสนับสนุน ภาษาไพธอนเป็นภาษาที่มีการเปิดเผยซอร์สโค้ด (Open source language) และสามารถดาวน์โหลดตัวแปรภาษาได้จาก www.python.org โดยไม่เสียค่าใช้จ่ายใดๆ

ภาษาไพธอนมีความยืดหยุ่นสูง สามารถนำมาใช้งานได้หลากหลาย และนำกลับมาใช้ใหม่ได้ ในโครงงานนี้ได้เลือกทำอินเตอร์พรีตเตอร์ขนาดเล็กสำหรับภาษาไพธอน เนื่องด้วยความสามารถดังที่ได้กล่าวมาแล้วทำให้มีความง่ายและสะดวกเป็นอย่างมากในการพัฒนาตัวภาษาเพื่อใช้กับบราวเซอร์ได้

2.1 คุณสมบัติของภาษาไพธอน

ภาษาไพธอนมีคุณสมบัติ และมีความสามารถหลายประการดังนี้

- **High-level**

ภาษาไพธอนมีโครงสร้างข้อมูลระดับสูงซึ่งช่วยลดระยะเวลาในการพัฒนาโปรแกรมลง เช่น Python's list (Resizeable arrays) และ Dictionary (Hash table) ซึ่งการสร้างสิ่งเหล่านี้ในภาษาอื่นๆ เช่น ในภาษา C ทำได้ยากเนื่องจากความจำเป็นที่จะต้องใช้โครงสร้างข้อมูล และ pointer ในภาษา C ข้อดีของการมีโครงสร้างข้อมูลระดับสูงคือ ลดระยะเวลาในการพัฒนาโปรแกรม, ขนาดของโปรแกรม และทำให้โปรแกรมอ่านง่ายซึ่งจะส่งผลให้การ debug โปรแกรมทำได้โดยง่าย

- **Object-oriented**

ภาษาไพธอนถูกสร้างมาเป็นภาษาเชิงวัตถุ(Object Oriented Language) เช่นเดียวกับ C++ และ JAVA และสามารถสร้างคลาสย่อยจากภาษาทั้งสองได้อีกด้วย จึงเหมาะแก่การเป็นภาษาสคริปต์ให้กับภาษาทั้งสอง

- **Scalable**

ภาษาไพธอนมีการเขียนซอร์สโค้ดที่เข้าใจง่าย มีโครงสร้างระดับสูง และสามารถสร้างแพ็คเก็จ (package) ที่บรรจุหลายๆคอมโพเน็น (component) ซึ่งสิ่งเหล่านี้สนับสนุนการออกแบบโปรแกรมเป็น

บล็อกของโปรแกรม และเมื่อต้องการจะขยายโปรแกรมโครงสร้างที่เป็นโมดูล และ pluggable ของไพธอนทำให้สามารถจัดการขยายได้โดยง่าย

- **Extensible**

การเขียนโปรแกรมในภาษาไพธอนสามารถแบ่งซอร์สโค้ดออกเป็นโมดูลย่อยซึ่งโมดูลย่อยเหล่านี้ และโมดูลใน standard library ของ ไพธอน ผู้เขียนโปรแกรมสามารถเรียกมาใช้ได้ นอกจากนี้ยังสามารถห่อ(wrapping) lower-level code โดยใช้ไพธอน interface ซึ่งทำให้สามารถสร้าง และ import โมดูลที่คอมไพล์แล้วเข้ามาใช้งานได้ โดยวิธีการเรียกใช้งานยังคงเหมือนโมดูลปกติในไพธอนแต่ประสิทธิภาพในการทำงานจะดีขึ้น ซึ่งความสามารถนี้ทำให้ไพธอนสามารถ extend โมดูลจากภาษาอื่นได้ด้วย เช่น ภาษาJava สำหรับ Jpython

- **Portable**

ภาษาไพธอนถูกพัฒนาขึ้นมาจากภาษา C โปรแกรมที่เขียนจากภาษาไพธอนจะสามารถใช้ได้กับระบบปฏิบัติการทั่วไป โดยตัวแปรภาษาจะแปรคำสั่งให้เป็น bytecode ที่ระบบปฏิบัติการนั้นเข้าใจ

- **Easy-to-learn**

ภาษาไพธอนมีคีย์เวิร์ดน้อย มีโครงสร้างภาษาที่เรียบง่าย และมี syntax ที่เข้าใจง่ายทำให้สามารถเรียนรู้ได้ง่าย และรวดเร็ว

- **Easy-to-read**

syntax ของภาษาไพธอนไม่มีการใช้สัญลักษณ์ที่มักพบในภาษาอื่นๆ ที่ใช้สำหรับในการเข้าถึงตัวแปร, การกำหนดสโคปของ Code block และการทำ pattern maching เช่น dollar sign($) ,semicolons(;), tildes(~) ซึ่งการที่ภาษาไพธอนหลีกเลี่ยงการกระทำดังกล่าวทำให้ซอร์สโค้ดของภาษาไพธอนสามารถอ่านทำความเข้าใจได้ง่าย

- **Easy-to-maintain**

เนื่องจากภาษาไพธอนมีคุณสมบัติ easy-to-learn และeasy-to-read ทำให้เกิดข้อดีคือ มีความซับซ้อนน้อยลง จึงทำให้การบำรุงรักษา และการพัฒนาต่อทำได้โดยง่าย

- **Robust**

ภาษาไพธอนมีการจัดการข้อผิดพลาดที่ดี โดยเมื่อไพธอนเกิดมีการทำงานที่ผิดพลาดขึ้นก็จะมี stack trace ซึ่งจะระบุว่าเกิดข้อผิดพลาดขึ้นที่ไหน และอย่างไรบ้าง และภาษาไพธอนยังมีความสามารถในการให้ผู้ใช้เขียนโปรแกรมเพื่อจัดการกับข้อผิดพลาด(Exception handler) อีกด้วย

- **Effective as a Rapid Prototyping Tool**

การเขียนโปรแกรมบนระบบใดๆโดยใช้ภาษาไพธอนเพียงอย่างเดียวสามารถทำได้ ถึงแม้ว่า compiled language อื่นๆ ก็สามารถทำได้เช่นกัน แต่ไพธอนมี library จำนวนมากให้ใช้ทำให้ลดระยะเวลาในการพัฒนาลงได้ ซึ่ง library ที่มีมาในตัวแปรภาษาอยู่แล้ว เช่น networking, Internet/Web/CGI, graphics and graphical user interface (GUI) development(Tkinter), imaging(PIL), numerical computation and analysis(Numpy), database access, hypertext(HTML,XML,SGML)

- **A Memory Manager**

ข้อเสียที่สำคัญของการเขียนโปรแกรมในภาษา C และC++ คือ การที่ผู้เขียนโปรแกรมต้องรับผิดชอบในการจัดการกับหน่วยความจำเอง แต่ในภาษาไพธอนตัวแปรภาษาจะทำหน้าที่นี้ให้โดยอัตโนมัติ ทำให้ผู้เขียนโปรแกรมไม่ต้องมากังวลกับการจัดการหน่วยความจำ ซึ่งส่งผลดีคือ ทำให้โปรแกรมมีข้อผิดพลาดน้อยลง และระยะเวลาในการพัฒนาน้อยลง

- **Interpreted and (Byte-) Compiled**

ไพธอนเป็นภาษาที่มีการทำงานแบบInterpreter ทำให้การทำงานช้ากว่าภาษาที่ใช้การ compile เนื่องจากการประมวลผลคำสั่งไม่ได้ทำในรูปแบบของภาษาเครื่องของระบบ(system's native binary language) แต่ภาษาไพธอนก็สามารถทำ byte-compiled ได้ซึ่งผลจากการ compile จะอยู่ในรูปแบบที่ใกล้เคียงกับภาษาเครื่องทำให้ประสิทธิภาพการทำงานดีขึ้น และยังมีความเป็นภาษาที่มีการทำงานแบบ Interpreter อยู่

	Execution Speed	Coding Speed	Object-Oriented	GUI Coding	Dev Environment	Suitability for large tasks	Libraries available
Python	Fair	Excellent	Excellent	Good	Fair	Excellent	Good
Perl	Fair	Excellent	Fair	Good	Fair	Fair	Excellent
Vis.Basic	Good	Excellent	Fair	Excellent	Excellent	Poor	Fair
C	Excellent	Poor	Poor	n/a	Excellent	Good	Good
C++	Excellent	Fair	Excellent	n/a	Excellent	Excellent	Good
Java	Fair	Good	Excellent	Good	Excellent	Excellent	Good

ตารางที่ 2-1 ตารางเปรียบเทียบคุณสมบัติต่างๆ ของภาษาไพธอนกับภาษาอื่นๆ

2.2 ชนิดของข้อมูล

ชนิดของข้อมูลในภาษาไพธอนจะอยู่ในอ๊อบเจ็ค (object) ที่ชื่อว่า ไพธอนอ๊อบเจ็ค การกำหนดชนิดของข้อมูลให้กับตัวแปรในภาษาไพธอนจะถูกกระทำโดยอัตโนมัติ ผู้เขียนโปรแกรมสามารถเปลี่ยนแปลงตัวแปรหนึ่งให้เก็บค่าที่ต่างไปได้โดยไม่ต้องสนใจชนิดของตัวแปรที่เก็บอยู่เดิม ชนิดของข้อมูลทั่วไปได้แก่

2.2.1 ตัวเลข (Numeric)

ข้อมูลชนิดตัวเลข ได้แก่

2.2.1.1 เลขฐานสิบ (Integer) คือเลขที่ประกอบไปด้วยเลข 0-9 ไม่ว่าจะเป็นจำนวนบวก หรือจำนวนลบแต่ต้องไม่มีจุดทศนิยม ตัวอย่างเช่น 1, -3, 8784 ฯลฯ

2.2.1.2 เลขฐานแปดคือ ตัวเลขที่ขึ้นต้นด้วย 0 เช่น 0715 ฯลฯ

2.2.1.3 เลขฐานสิบหกคือ ตัวเลขที่ขึ้นต้นด้วย 0x เช่น 0x9FAC, 0x12A ฯลฯ

2.2.1.4 เลขทศนิยม(Floats) เช่น 3.2, -45e12 ฯลฯ

2.2.1.5 เลขจำนวนเต็มขนาดยาว(Long integer) คือตัวเลขที่ลงท้ายด้วยตัว L หรือ l เช่น 4890549579L, 5l ฯลฯ

2.2.1.6 เลขจำนวนเชิงซ้อน(Complex number) เช่น 9j, 5i-4j, 45i ฯลฯ

2.2.2 จำนวนทางตรรกะ (Boolean)

โดยปกติจำนวนทางตรรกะจะมีค่าเท็จ หรือจริง สำหรับภาษาไพธอนจะแทนค่าเท็จด้วย เลขศูนย์, โครงสร้างเปล่า หรือ None value เช่น 0, [], {}, (), None และแทนค่าจริงด้วย ค่าใดๆที่ไม่ใช่ศูนย์, โครงสร้างที่มีข้อมูลอยู่ เช่น 1, [5], (7,8,98,347), “xyz”

2.2.3 สายอักขระ (String)

ข้อมูลชนิดสายอักขระ คือ ข้อมูลชนิดของตัวอักษร เช่น ‘This is a string.’, “This is another string” สายอักขระนี้จะอยู่ภายในสัญลักษณ์ Single quoted หรือ Double quoted ก็ได้ แต่ต้องเหมือนกันทั้งหัว และท้าย หากใช้ Single quoted ในสายอักขระจะสามารถมี Double quoted ได้ และในทำนองเดียวกันหากใช้ Double quoted ก็จะสามารถมี Single quoted ในสายอักขระได้เช่นเดียวกัน

ในข้อมูลชนิดสายอักขระนั้นจะมีอักขระพิเศษอยู่ด้วย ได้แก่

\n = Newline \’ = Single quoted\b = Backspace

\t = Tab \” = Double quoted \f = Formfeed

\\ = Backslash \a = Bell \r = Carriage return

\v = Vertical tab

สำหรับการใช้ ‘’’ หรือ “”” นั้นจะเป็นสายอักขระที่ หากมีการขึ้นบรรทัดใหม่ในสายอักขระ หรือมีการย่อหน้าจะทำการเปลี่ยนให้เป็นตัวอักขระพิเศษ

2.2.4 ลิสต์ (Lists)

ข้อมูลชนิดลิสต์นี้จะเป็นการนำเอาข้อมูลชนิดอื่นๆมาเก็บไว้ใยข้อมูลชนิดนี้ โดยจะมีลักษณะคล้ายกับข้อมูลชนิดอาเรย์(Array) แต่จะต่างกันตรงที่ข้อมูลที่เก็บในลิสต์ไม่จำเป็นต้องเก็บข้อมูลชนิดเดียวกัน ข้อมูลที่สามารถเก็บในลิสต์ได้แก่ ตัวเลข, ตัวอักษร, สายอักขระ หรือแม้แต่จะเป็นลิสต์ด้วยกันเองก็ได้

การเขียนอ้างอิงถึงข้อมูลชนิดลิสต์ทำได้ดังนี้

list1 = []	เป็นการสร้างลิสต์เปล่า
list2 = [1,"two",[3,4]]	เป็นการสร้างลิสต์ที่ประกอบด้วยสมาชิก 3 ตัวโดยตัวแรกเป็น ตัวเลข, ตัวที่สองเป็นสายอักขระ, ตัวที่สามเป็นลิสต์ที่มีสมาชิกเป็นตัวเลขสองตัว

การอ้างถึงสมาชิกในลิสต์ทำได้ดังนี้

list2[0]	จะได้ผลลัพธ์เป็น 1
list2[1:2]	จะได้ผลลัพธ์เป็น "two"
list2[1:]	จะได้ผลลัพธ์เป็น ["two",[3,4]]
list2[-1]	จะได้ผลลัพธ์เป็น [3,4]

2.2.5 ทัปเปิล (Tuples)

ข้อมูลชนิดทัปเปิลมีลักษณะเหมือนข้อมูลชนิดลิสต์ แต่จะต่างกันตรงที่ไม่สามารถแก้ไขข้อมูลของสมาชิกภายในทัปเปิลได้ ทัปเปิลจึงเหมาะแก่การใช้เป็นข้อมูลอ้างอิง การกำหนดทัปเปิลมีวิธีการดังนี้

tuple1 = ()	เป็นการสร้างทัปเปิลเปล่า
tuple2 = ("one",2,"three",4)	

การจัดการกับทัปเปิลนั้นมีวิธีการเช่นเดียวกับการจัดการกับลิสต์

2.2.6 ดิกชันนารี (Dictionary)

ข้อมูลชนิดดิกชันนารีจะประกอบไปด้วยคีย์ (Keys) และค่าที่เก็บ (Values) ในการอ้างอิงข้อมูลประเภทดิกชันนารีทำได้ดังนี้

dic[key]	จะให้ผลลัพธ์เป็นอ๊อบเจ็คที่ถูกเก็บโดยคีย์นั้น
dic.has_key(key)	จะส่งค่ากลับมาเป็น 1 เมื่อมีคีย์ที่ระบุอยู่ใน dic และ 0 เมื่อไม่มีคีย์ที่ระบุอยู่ใน dic
dic.keys()	ส่งค่ากลับมาเป็นลิสต์ของคีย์ต่างๆที่มีอยู่ใน dic
dic.values()	ส่งค่ากลับมาเป็นลิสต์ของค่าที่เก็บต่างๆที่มีอยู่ใน dic
dic.clear()	ทำการลบทุกoค่าที่มีอยู่ใน dic

โดยปกติข้อมูลชนิดลิสต์จะสามารถเรียงลำดับข้อมูลได้ แต่ข้อมูลชนิดดิกชันนารีนั้นจะไม่มีการเรียงลำดับข้อมูล การอ้างอิงทำได้โดยผ่านทางคีย์เท่านั้น

2.2.7 ข้อมูลเปล่า (None)

ข้อมูลชนิดนี้ใช้ในการระบุค้าเริ่มต้นของตัวแปร หรือแสดงว่าไม่มีข้อมูล ซึ่งในการเปรียบเทียบค่านั้น None จะมีค่าเท่ากับ None เท่านั้น

2.3 ตัวกระทำ (Operators)

ตัวกระทำต่างๆแบ่งออกเป็น 4 ประเภทคือ ตัวกระทำทางตรรกะ, ตัวกระทำทางการเปรียบเทียบ, ตัวกระทำทางบิตไวส์ และตัวกระทำทางคณิตศาสตร์

2.3.1 ตัวกระทำทางตรรกะ (Logical operators)

ตัวกระทำทางตรรกะมีทั้งหมด 3 ตัวด้วยกันคือ and, or, not โดยทั้ง 3 ตัวกระทำนี้สามารถใช้งานได้ด้วยการเขียน and,or และ not โดยตรง

2.3.2 ตัวกระทำทางการเปรียบเทียบ (Comparison operators)

ตัวกระทำที่ใช้เปรียบเทียบค่าต่างๆมีดังต่อไปนี้

<	น้อยกว่า	>	มากกว่า
<=	น้อยกว่าหรือ เท่ากับ	>=	มากกว่าหรือ เท่ากับ
==	เท่ากับ	<>,!=	ไม่เท่ากับ(สามารถใช้ได้ทั้ง 2 แบบ)
in	x in y หมายความว่ามีค่า x เป็นสมาชิกใน y หรือไม่		
not in	ให้ผลตรงข้ามกับ in		
is	x is y หมายความว่า x เป็นสิ่งเดียวกับ y ซึ่งไม่เหมือนกับ ==		
is not	ให้ผลตรงข้ามกับ is		

2.3.3 ตัวกระทำทางบิตไวส์ (Bitwise operators)

<<	integer<< n ส่งค่ากลับเป็นตัวเลขที่เกิดจากการเลื่อนบิตของ integer ไปทางซ้าย (Shift left)ไปเป็นจำนวน n ตัว
>>	ทำงานเช่นเดียวกับ << แต่เลื่อนไปทางขวา(Shift right)
&	m&n ผลลัพธ์คือการ and กันในรูปตัวเลขฐานสองของ m และ n
|	m|n ผลลัพธ์คือการ or กันในรูปตัวเลขฐานสองของ m และ n
^	m^n ผลลัพธ์คือการ xor กันในรูปตัวเลขฐานสองของ m และ n
~	~m ผลลัพธ์คือการ not ในรูปตัวเลขฐานสองของ m

2.3.4 ตัวกระทำทางคณิตศาสตร์ (Arithmatic-Style operators)

ตัวกระทำทางคณิตศาสตร์นี้สามารถใช้งานไม่เพียงกับตัวเลขเท่านั้น แต่ยังสามารถใช้งานกับข้อมูลชนิดอื่นๆ เช่นสายอักขระได้ ตัวกระทำดังกล่าวได้แก่

+	หากใช้กับตัวเลขจะเป็นการบวกกัน แต่หากใช้กับสายอักขระจะเป็นการนำมาเรียงต่อกัน และหากเป็นลิสต์ หรือทัปเปิลจะเป็นการรวมสมาชิกของสองลิสต์เข้าเป็นลิสต์เดียว หรือสมาชิกของสองทัปเปิลเป็นทัปเปิลเดียว
-	ใช้กับตัวเลขเพื่อทำการลบกัน
*	หากใช้กับตัวเลขจะเป็นการคูณกัน แต่หากใช้กับสายอักขระ, ลิสต์ หรือทัปเปิลจะเป็นการเพิ่มจำนวนของข้อมูลนั้น เช่น 'M'*5 จะได้ 'MMMMM' เป็นต้น
/	ใช้กับตัวเลขเพื่อทำการหาร โดยหากใช้กับจำนวนเต็มผลที่ได้ก็จะเป็นจำนวนเต็ม เศษที่ได้จะถูกปัดลง ตัวกระทำนี้เหมือนกับ div ในภาษา Pascal
**	ใช้กับตัวเลขเท่านั้นเป็นการยกกำลัง
%	หากใช้กับตัวเลขจะเป็นการหาเศษจากการหาร เหมือนการใช้คำสั่ง mod ในภาษา Pascal หากใช้กับสายอักขระ จะเป็นการบอกรูปแบบในการพิมพ์

2.3.5 ลำดับของตัวกระทำ (Precedence)

ลำดับของตัวกระทำมีลำดับดังนี้

or

and

not

<, <=, ==, >=, >, !=, <>, is, in, not, not in

|

^

&

<<, >>

+, -

*, /, %

**

unary+, unary-, unary~

2.4 คำสงวน (Reserved words)

คำสงวนในภาษาไพธอนแบ่งเป็น คีย์เวิร์ด(Keywords) และบิวท์อินฟังก์ชั่น(Built-in function)

คีย์เวิร์ด ได้แก่

'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'

บิวท์อินฟังก์ชั่นได้แก่

'__import__', 'abs', 'apply', 'buffer', 'callable', 'chr', 'cmp', 'coerec', 'compile', 'complex', 'delattr', 'dir', 'divmod', 'eval', 'execfile', 'filter', 'float', 'getattr', 'globals', 'hasattr', 'hash', 'hex', 'id', 'input', 'int', 'intern', 'isins', 'issubclass', 'len', 'list', 'locals', 'long', 'map', 'max', 'min', 'oct', 'open', 'ord', 'pow', 'range', 'raw_input', 'reduce', 'reload', 'repr', 'round', 'setattr', 'slide', 'str', 'tuple', 'type', 'vars', 'xrange'

2.5 กลุ่มของโค้ด (Code block)

การกำหนดสโคปของซอร์สโค้ดใช้การย่อหน้า(Indentation) แทนสัญลักษณ์ ซึ่งต่างจากภาษาอื่นๆ เช่น ภาษาC และ Java ใช้วงเล็บปีกกา และภาษา Pascal ใช้ begin end เป็นตัวกำหนดสโคปของซอร์สโค้ด

การที่ไพธอนใช้การย่อหน้าในการกำหนดสโคปของซอร์สโค้ดทำให้ช่วยลดความยุ่งยาก และข้อผิดพลาดที่มักเกิดขึ้นจากการลืมพิมพ์สัญลักษณ์ที่ใช้ในการกำหนดสโคป และข้อดีอีกอย่างหนึ่งคือ ทำให้ซอร์สโค้ดอ่านง่ายเนื่องจากทุกครั้งที่ขึ้นสโคปใหม่จะต้องทำการย่อหน้าซึ่งทำให้ซอร์สโค้ดมีความเป็นระเบียบ

2.6 คอมเมนต์ (Comment)

สำหรับการคอมเมนต์ นั้นจะใช้ '#' เป็นตัวระบุ โดยข้อความที่อยู่หลัง '#' ไปจนสุดบรรทัดจะไม่ถูกประมวลผลโดยตัวแปลภาษาไพธอน

2.7 การพิมพ์ statement เดียวในหลายบรรทัด (Continuation)

เนื่องจาก statement ในภาษาไพธอนจะถูกแบ่งโดยการขึ้นบรรทัดใหม่(NEWLINE) ดังนั้นหนึ่ง statement จะต้องใช้หนึ่งบรรทัด แต่ว่าสามารถใช้เครื่องหมาย backslash(\) ในการพิมพ์ statement เดียวในหลายบรรทัดได้ดังนี้

```
if (weather_is_hot == 1) and \
   (available == 1):
        goto_the_beach()
```

2.8 การพิมพ์หลาย statement ในบรรทัดเดียว

ใช้เครื่องหมาย semicolon(;) ในการพิมพ์หลาย statement ในบรรทัดเดียว เช่น

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

2.9 การกำหนดค่าให้ตัวแปร (Variable assignment)

การกำหนดค่าให้ตัวแปรใช้เครื่องหมายเท่ากับ (=) ซึ่งการกำหนดค่าให้ตัวแปรนั้นไม่ได้เป็นการนำค่าไปใส่ให้ตัวแปรจริงๆ แต่เป็นการกำหนดให้ตัวแปรอ้างอิงไปยังไพธอนอ๊อบเจ็คที่มีค่านั้นอยู่ โดยไพธอนอ๊อบเจ็คนั้นอาจถูกสร้างขึ้นมาใหม่ตอนที่กำหนดค่า หรือเป็นไพธอนอ๊อบเจ็คที่มีอยู่แล้วจากตัวแปรอื่นก็ได้

2.9.1 การกำหนดค่าหนึ่งค่าให้กับหลายตัวแปร (Multiple assignment)

x = y = z = 1

ตัวอย่างด้านบนเป็นการกำหนดค่าหนึ่งค่าให้กับหลายตัวแปร โดยอ๊อบเจ็คชนิดจำนวนเต็มที่มีค่า 1 จะถูกสร้างขึ้น และตัวแปร x, y, z จะถูกกำหนดให้อ้างอิงไปยังอ๊อบเจ็คนี้ ซึ่งนี่คือขั้นตอนการกำหนดค่าหนึ่งอ๊อบเจ็คให้กับหลายตัวแปร

2.9.2 การกำหนดค่าหลายค่าให้กับหลายตัวแปร

x, y, z = 1, 2, 'three'

ตัวอย่างด้านบนเป็นการกำหนดค่าหลายค่าให้กับหลายตัวแปรโดยผลลัพธ์ที่ได้คือ x, y และ z มีค่าเท่ากับ 1, 2 และ 'three' ตามลำดับ การกำหนดค่าแบบนี้ทั้งสองข้างของเครื่องหมายเท่ากับคือทัปเปิล ซึ่งตัวอย่างด้านบนสามารถเขียนได้อีกอย่างคือ (x, y, z) = (1, 2, 'three')

การกำหนดค่าโดยใช้วิธีนี้ยังมีประโยชน์อีกอย่างในกรณีที่ต้องการสลับค่าตัวแปรซึ่งสามารถเขียนเพียง statement เดียวก็สามารถสลับค่าตัวแปรได้แล้ว เช่น

x, y = y, x หรือ (x, y) = (y, x)

แต่ถ้าใช้ภาษาอื่น เช่น ภาษา C ต้องเขียนดังนี้

```
temp = x
x = y
y = temp
```

2.10 คำสั่งควบคุมการทำงาน (Control flow)

คำสั่งลูปเป็นคำสั่งที่ใช้ในการควบคุมลำดับการประมวลผลของคำสั่งต่างๆ โดยให้ประมวลผลวนรอบไปเรื่อยๆจนกว่าจะถึงเงื่อนไขที่กำหนด คำสั่งลูปต่างๆได้แก่ IF, WHILE, FOR

2.10.1 คำสั่ง IF

รูปแบบของคำสั่ง IF มีดังนี้

```
If <EXPRESSION>:
        <STATEMENT>
elif <EXPRESSION>:
```

```
        <STATEMENT>
else:
        <STATEMENT>
```

โดย <EXPRESSION> นั้นหมายถึงเงื่อนไขในการหยุดลูป ส่วน <STATEMENT> คือคำสั่งที่อยู่ในลูป โดยตัวอย่างของการใช้คำสั่ง IF เช่น

```
if  x == 0:
        print 'x equal 0'
elif x >0:
        print 'x is positive number'
else:
        print 'x is negative number'
```

2.10.2 คำสั่ง WHILE

คำสั่ง WHILE นั้นมีการตรวจสอบเงื่อนไขก่อนการทำคำสั่งในลูปทุกครั้ง โดยมีรูปแบบคือ

```
while <EXPRESSION> :
        <STATEMENT>
```

ตัวอย่างคำสั่ง WHILE เช่น

```
while 1:
        Print 'Infinite loop'
```

2.10.3 คำสั่ง FOR

คำสั่ง FOR เป็นคำสั่งที่จะวนรอบการทำงานจนครบจำนวนที่กำหนด รูปแบบของคำสั่งคือ

```
For <VARIABLE> in <RANGE>:
        <STATEMENT>
```

โดย <RANGE> คือตัวแปรที่เป็นลิสต์ หรือคำสั่งเฉพาะบางคำสั่ง ส่วน <VARIABLE> คือตัวแปรที่เก็บค่าของสมาชิกทุกตัวของ<RANGE> การทำงานของคำสั่ง FOR จะทำงานวนรอบเป็นจำนวนเท่ากับจำนวนสมาชิกของ <RANGE> ตัวอย่างเช่น

```
list1 = [1,'a',[b,c]]
for var1 in list1:
        print var1
```

จากซอร์สโค้ดข้างต้นได้ผลลัพธ์ดังนี้

```
1
'a'
[b,c]
```

การทำงานของคำสั่ง FOR ข้างต้นจะทำงานเป็นจำนวน 3 ครั้ง เท่ากับจำนวนสมาชิกใน list1

```
for i in range(8):
    print i
```

คำสั่ง range(n) จะส่งค่ากลับเป็นตัวเลขตั้งแต่ 0 จนถึง n-1 ผลลัพธ์ที่ได้คือ

```
0
1
2
3
4
5
6
7
```

2.11 การจัดการหน่วยความจำ (Memory management)

ลักษณะเด่นของตัวแปร และการจัดการหน่วยความจำที่ผู้ใช้งานสามารถสังเกตได้มีดังนี้

- ตัวแปรจะถูกประกาศตอนให้ค่าครั้งแรก
- ไม่มีการกำหนดชนิดของตัวแปร
- ผู้เขียนโปรแกรมไม่ต้องเขียนโปรแกรมจัดการกับหน่วยความจำ
- ชื่อตัวแปรที่เคยประกาศใช้ไปแล้วสามารถนำมาใช้ใหม่ได้ถึงแม้ว่าจะมีชนิดของตัวแปรที่ต่างกัน
- สามารถใช้คำสั่ง del เพื่อทำการปลดปล่อยหน่วยความจำอย่างเปิดเผยในการเขียนโค๊ดได้

2.11.1 การประกาศตัวแปร (Variable Declarations)

ตัวแปรจะถูกประกาศตอนให้ค่าครั้งแรกด้วยโอเปอเรเตอร์เท่ากับ(Assignment operation) โดยไม่ต้องมีการกำหนดชนิดของตัวแปร

2.11.2 การจองหน่วยความจำ (Memory Allocation)

ผู้เขียนโปรแกรมไม่ต้องจัดการเกี่ยวกับการจอง และการปลดปล่อยหน่วยความจำ เนื่องจากไพธอนจะจัดการกับการทำงานในระดับต่ำ (Lower-level) เช่นนี้ให้อยู่แล้ว

2.11.3 การรวบรวมหน่วยความจำที่ไม่ได้ใช้ (Garbage Collection)

หน่วยความจำที่ไม่ได้ใช้ ระบบจะมีกลไกในการรวบรวมหน่วยความจำเหล่านี้กลับมาใช้ใหม่ โดยที่ผู้เขียนโปรแกรมไม่ต้องเขียนโค๊ดในส่วนนี้เนื่องจากไพธอนจะใช้การนับการอ้างอิง(Reference counting) ในการตัดสินว่าหน่วยความจำส่วนไหนที่ไม่ได้ใช้อีกต่อไปอีกแล้ว

2.11.4 การนับการอ้างอิง (Reference counting)

ไพธอนจะมีตัวแปรที่ใช้สำหรับนับการอ้างอิงที่กระทำไปยังไพธอนอ๊อบเจ็ค โดยค่าเริ่มต้นของตัวแปรนี้คือ หนึ่ง เมื่อตอนประกาศตัวแปร และเมื่อไพธอนอ๊อบเจ็คถูกอ้างอิงอีก หรือถูกใช้เป็นอากิวเมนต์ในการเรียกใช้ฟังก์ชั่น หรือเมธอด ค่านี้ก็จะถูกเพิ่มขึ้นอีกหนึ่ง และเมื่อไรที่ค่านี้มีค่าเป็นศูนย์ก็จะหมายถึงไม่มีการอ้างถึงอ๊อบเจ็คนี้อีกแล้ว และหน่วยความจำในส่วนนี้ก็จะถูกนำกลับมาใช้งานใหม่

```
foo1 = 'foobar'          # reference count = 1
foo2 =  foo1             # reference count = 2
Check_value(foo1)        # reference count = 3 ตอนเรียกใช้ และหลังจากใช้เสร็จจะมาเท่าเดิม
```

จากตัวอย่างด้านบนเมื่อประกาศตัวแปรครั้งแรก reference count ของสายอักขระ 'foobar' จะมีค่าเท่ากับหนึ่ง และเมื่อถูกอ้างอิงโดย foo2 ค่า reference count ก็จะถูกเพิ่มเป็นสอง และในตอนที่เรียกใช้ฟังก์ชั่นก็จะถูกเพิ่มเป็นสาม แต่พอจบจากฟังก์ชั่นก็จะถูกลดไปเป็นสอง และถ้าใช้คำสั่ง del กับ foo2 ค่า reference count ก็จะถูกลดเป็นหนึ่ง และเมื่อ foo1 หลุดออกจาก scope ค่า reference count ก็จะถูกลดลงเหลือศูนย์ และหน่วยความจำในส่วนนั้นก็จะถูกนำกลับมาใช้งานใหม่

2.11.5 คำสั่ง del

คำสั่ง del มีรูปแบบดังนี้

```
del obj
```

ตัวอย่างเช่น del foo2 จะทำให้เกิดผลสองอย่างดังนี้คือ

- ลบชื่อ foo2 ออกจาก namespace
- ลดค่า reference count ของอ๊อบเจ็คที่มันอ้างถึง ('foobar') ลงหนึ่ง

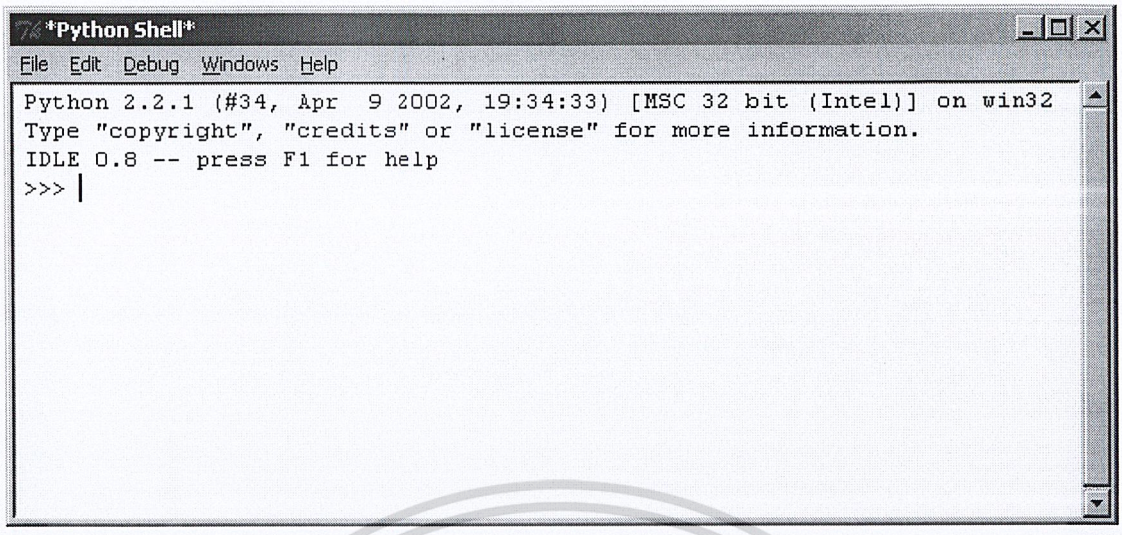
2.11.6 การลดค่า reference count

ค่า reference count จะถูกลดลงเมื่อเกิดเหตุการณ์ต่อไปนี้

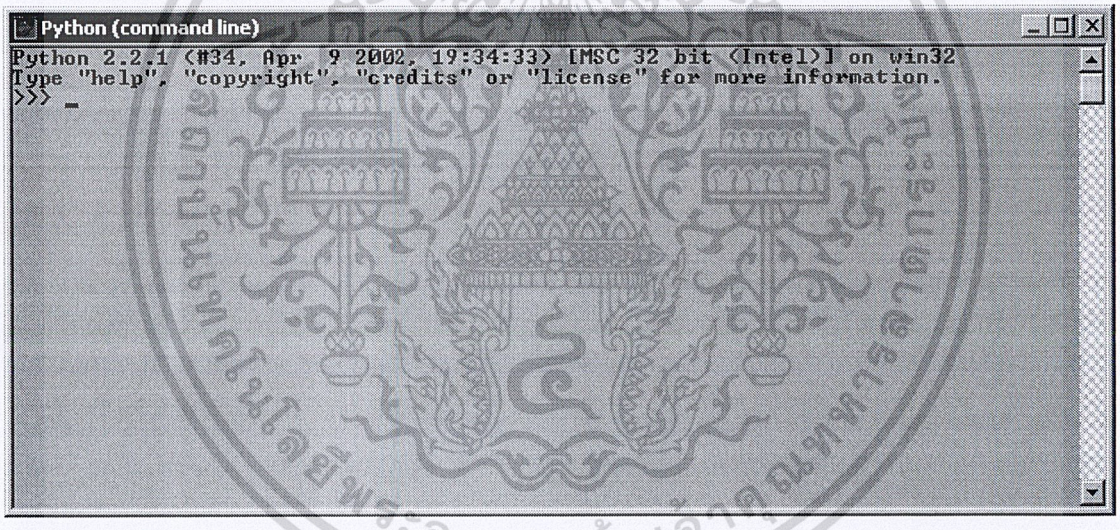
- ใช้คำสั่ง del
- ตัวแปรถูกเปลี่ยนการอ้างอิงจากอ๊อบเจ็คเดิมไปยังอ๊อบเจ็คใหม่ ซึ่งอ๊อบเจ็คเดิมนั้นก็จะถูกลดค่า reference count ลงหนึ่ง
- หลุดออกจาก scope

2.12 การใช้งานภาษาไพธอน

ภาษาไพธอนที่โครงงานเลือกใช้คือภาษาไพธอนเวอร์ชั่น 2.2.1 ในชุดของโปรแกรมจะประกอบไปด้วย คู่มือการใช้งานเป็นเอกสาร HTML, และโปรแกรมหน้าจอโต้ตอบซึ่งมีให้เลือกใช้ 2 อย่างคือ IDLE ซึ่งเป็นไพธอนGUI และไพธอนคอมมานด์ไลน์ซึ่งเป็นหน้าจอโต้ตอบอยู่บน DOS



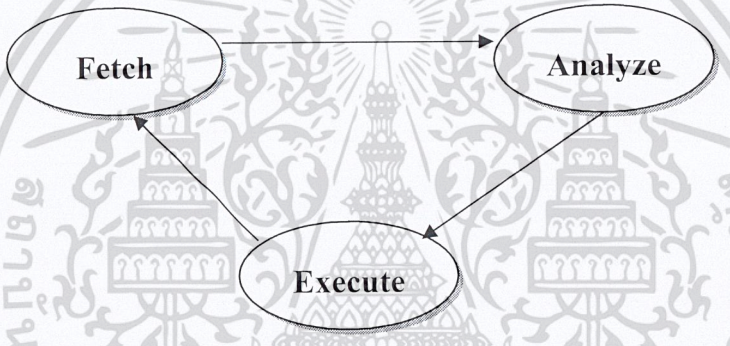
รูปที่ 2-1 แสดงโปรแกรมไพธอน IDLE



รูปที่ 2-2 แสดงโปรแกรมไพธอนคอนโซล

บทที่ 3
อินเตอร์พรีตเตอร์

อินเตอร์พรีตเตอร์เป็นโปรแกรมที่ทำการอ่านโปรแกรมหนึ่งๆ ที่เขียนขึ้นจากภาษาใดๆ ซึ่งเรียกว่าซอร์สโค้ด แล้วทำการแปลความหมายของโค้ดนั้นที่ละหนึ่งชุดคำสั่ง หรือที่ละหนึ่งการกระทำ เพื่อนำไปกระทำตามความหมายของคำสั่งนั้นๆ โดยการทำงานทั่วๆ ไปของอินเตอร์พรีตเตอร์นั้นจะมีอยู่ด้วยกัน 3 ส่วนใหญ่ๆ ดังรูปต่อไปนี้



รูปที่ 3-1 แสดงส่วนประกอบหลักของอินเตอร์พรีตเตอร์

3.1 การเฟ็ชคำสั่ง (Fetch)

การเฟ็ชคำสั่งนั้น คือการอ่านข้อมูลจากซอร์สโค้ดเข้ามายังตัวโปรแกรมอินเตอร์พรีตเตอร์ โดยที่ส่วนของการเฟ็ชคำสั่งนั้นจะต้องมีการทำการเช็ค ว่าควรจะอ่านมาเท่าใดซึ่งในหลักการแล้ว คือจะต้องทราบให้ได้ว่าบรรทัดที่เฟ็ชเข้ามานั้นได้ข้อมูลมาครบหนึ่งคำสั่งหรือไม่ หากยังไม่ครบจำเป็นที่จะต้องนำส่วนที่เหลือออกมาให้ครบเป็น 1 ชุดคำสั่ง เช่น

```
x = 10
```

จากตัวอย่างจะเห็นว่าหากเราอ่านข้อมูลเข้ามาจากบรรทัดแรกมาเพียงบรรทัดเดียวนั้นก็ถือว่าเพียงพอแล้ว เนื่องจาก x = 10 นั้นถือเป็นคำสั่ง 1 คำสั่งแล้ว แต่ถ้าหากเป็นดังตัวอย่างข้างล่างนี้

```
if x < 10 :
    y = 100
```

เราจะต้องทราบให้ได้ว่า การเฟ็ชแค่บรรทัดเดียวนั้นไม่เพียงพอที่จะนำข้อมูลนั้นไปวิเคราะห์ได้ จำเป็นที่จะต้องทำการอ่านข้อมูลมาอีก 1 บรรทัดเพื่อให้ได้ครบเป็น 1 ชุดคำสั่ง

3.2 การวิเคราะห์ข้อมูล (Analyze)

การวิเคราะห์ข้อมูลนั้นจะเป็นการนำข้อมูลที่ได้ทำการเฟ็ชเข้ามาทำการตรวจจับว่าคำต่างๆ ที่นำเข้ามานั้นถูกต้องตามรูปแบบของภาษานั้นๆ หรือไม่ (Lexical Analysis) จากนั้นหากพบว่าคำแต่ละคำถูกต้องตามรูปแบบที่กำหนดไว้แล้วก็จะทำการนำคำเหล่านั้นมามองเป็นรูปประโยคแล้วตรวจสอบว่าประโยคที่ได้จากคำต่างๆ มารวมกันนั้นถูกต้องตามรูปประโยคของภาษานั้นๆ (Syntax Analysis) หรือไม่ และเมื่อพบว่าประโยคนั้นๆ ถูกต้องสมบูรณ์แล้วก็จะนำประโยคที่ได้ไปแปลความหมายว่าชุดคำสั่งนั้นเป็นการกระทำใดเพื่อนำไปสร้างโครงสร้างของคำสั่งให้อยู่ในรูปแบบของทรี (Tree) ซึ่งเรียกกันว่าพาร์ซทรี (Parse Tree) เพื่อส่งต่อให้ส่วนการกระทำตามคำสั่งนั้นๆ อีกที

3.2.1 การตรวจจับคำ (Lexical Analysis)

เป็นการตรวจจับกลุ่มของสตริงค์ (String) เพื่อให้ทราบว่าสตริงค์ต่างๆ เหล่านั้นเป็นกลุ่มคำที่มีอยู่ในภาษาที่เราต้องการจะทำการอินเตอร์พรีตหรือไม่ (ซึ่งในที่นี้จะหมายถึงภาษาไพธอน) โดยกลุ่มคำดังกล่าวนั้นจะต้องที่การกำหนดขึ้นมาโดยการใช้เรกกูล่าร์เอ็กซ์เพรสชั่น (Regular Expression) สำหรับกำหนดรูปแบบของตัวกลุ่มคำ

เรกกูล่าร์เอ็กซ์เพรสชั่น คือ การการหนดกลุ่มของสตริงค์โดยใช้สัญลักษณ์ต่างๆ ที่เป็นมาตราฐานในการกำหนดความหมายต่างๆ ให้กับสตริงค์ โดยจะมีอยู่หลักๆ ดังต่อไปนี้

'|'	เลือกเอาไอเท็มตัวใดตัวหนึ่งที่ขั้นกลางด้วยสัญลักษณ์นี้
'*'	กำหนดว่าไอเท็มนั้นจะมีได้ตั้งแต่ 0 ตัวถึงเท่าไรก็ได้
'+'	กำหนดว่าไอเท็มนั้นจะมีได้ตั้งแต่ 1 ตัวขึ้นไป
'?'	กำหนดว่าไอเท็มนั้นจะมีได้ตั้งแต่ 0 หรือ 1 ตัว
'(' และ ')'	กำหนดกลุ่มไอเท็ม

ตัวอย่าง เรกกูล่าร์เอ็กซ์เพรสชั่นสำหรับกลุ่มคำ (Token) ของภาษาไพธอนจะออกมามีลักษณะดังต่อไปนี้

ชื่อ	เร็กกูล่าร์เอ็กซ์เพรสชั่น
identifier	({letter}|"_")({letter}|{digit}|"_")*
letter	({lowercase}|{uppercase})
lowercase	[a-z]
uppercase	[A-Z]
digit	[0-9]
integer	({decimalinteger}|{octinteger}|{hexinteger})
longinteger	{integer}[lL]
intpart	{digit}+
fraction	"."{digit}+
exponent	[eE][+-]?{digit}+
decimalinteger	{nonzerodigit}{digit}*|0
octinteger	0{octdigit}+
hexinteger	0[xX]{hexdigit}+
nonzerodigit	[1-9]
octdigit	[0-7]
hexdigit	({digit}|[a-f]|[A-F])

ตารางที่ 3-1 ตารางตัวอย่างเรกกูล่าร์เอ็กซ์เพรสชั่นของภาษาไพธอน

3.3 การตรวจสอบประโยค (Systax Analysis)

เป็นการตรวจสอบประโยคที่ได้มาจากกลุ่มคำที่ถูกต้องว่าเป็นประโยคที่ถูกต้องตามรูปแบบที่กำหนดไว้หรือไม่ ซึ่งในการที่จะกำหนดรูปแบบของประโยคต่างๆ ขึ้นมาได้นั้นจะต้องมีการกำหนดขึ้นโดยการใช้บีเอ็นเอฟ (BNF : Backus-Naur Form)

บีเอ็นเอฟ นั้นจะเป็นการกำหนดกฎหรือตั้งกฎ (rule) ของประโยคต่างๆ ของภาษาใดภาษาหนึ่งที่ต้องการสร้างประโยค (ซึ่งในที่นี้หมายถึงภาษาไพธอน) โดยลักษณะของ บีเอ็นเอฟจะมีลักษณะดังนี้

<non terminal> → <non terminal> หรือ <terminal> | … | <non terminal> หรือ <terminal>

3.3.1 non terminal

non terminal คือ สัญลักษณ์ที่ใช้แสดงถึงจุดที่ยังไม่สิ้นสุดของประโยคแต่สามารถที่จะขยายออกไปอีกได้โดยใช้กฎของ บีเอ็นเอฟ ตัวอื่นๆ ต่อไป เช่น

A → **B | C**

B → "x"

C → "y"

จากตัวอย่างข้างต้น ถ้าสมมติให้ ตัวอักษรตัวหนาหมายถึง non terminal ก็จะเห็นได้ว่าจาก บีเอ็นเอฟ บรรทัดแรกนั้น สามารถขยายต่อไปยัง บีเอ็นเอฟบรรทัดที่สองและบรรทัดที่สามได้อีกจนกว่าจะไปเจอส่วนที่เป็น terminal ก็จะไม่สามารถขยายต่อได้อีกนั่นเอง

3.3.2 terminal

terminal คือ จุดสิ้นสุดของกฎในบีเอ็นเอฟ ซึ่งอาจหมายถึงคำใดๆ หรือตัวอักษรใดๆ รวมทั้งกลุ่มคำที่ได้มาจากการกำหนดเรกกูล่าร์เอ็กซ์เพรสชั่นในขั้นตอนตรวจจับกลุ่มคำก็ได้

ตัวอย่าง บีเอ็นเอฟ สำหรับภาษาไพธอนจะมีลักษณะดังต่อไปนี้

ชื่อ	บีเอ็นเอฟ
primary	atom | target
atom	literal | enclosure
target	IDENTIFIER | subscription
literal	INTEGER | STRING | LONGINTEGER
sSubscription	primary '[' expression_list ']'
enclosure	'(' expression_list_ ')' | '[' expression_list_ ']'

ตารางที่ 3-2 ตารางตัวอย่างบีเอ็นเอฟของภาษาไพธอน

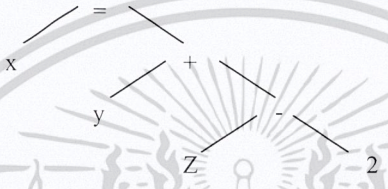
3.4 พาร์ซทรี (Parsetree)

พาร์ซทรี นั้นจะเป็นโครงสร้างของประโยค ที่ผ่านการตรวจสอบจากขั้นตอนที่ผ่านมาเรียบร้อยแล้ว โดยเราจะทำการแยกกลุ่มคำแต่ละตัวของประโยคออกมาสร้างเป็นลักษณะของทรี ซึ่งโครงสร้างดังกล่าวนี้เราจะเรียกว่า Abstract Syntax Tree (AST)

โดยโครงสร้างของคำสั่งแต่ละคำสั่งนั้นก็อาจจะมีลักษณะที่แตกต่างกันออกไปตามแต่ลักษณะของคำสั่ง โดยจะพิจารณาเอากลุ่มคำที่แสดงถึงการกระทำเป็นราก (root) ของทรีส่วนลูกทางด้านซ้าย (left child) และลูกทางด้านขวา (right child) นั้นก็จะเป็นส่วนของกลุ่มคำที่เป็นตัวถูกกระทำ โดยการสร้างพาร์ซทรีนั้นจะทำการตรวจสอบและสร้างจากประโยคซึ่งจะมองจากซ้ายไปขวา และความสำคัญของการกระทำเป็นหลัก

ตัวอย่าง พาร์ซทรีของคำสั่งทั่วๆ ไปในภาษาไพธอน เป็นดังต่อไปนี้

fetch input : x = y + Z - 2

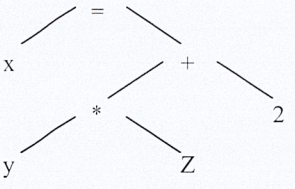


รูปที่ 3-2 แสดงพาร์ซทรีของคำสั่งทั่วไปของภาษาไพธอน

จากรูปจะเห็นได้ว่าเราทำการสร้างพาร์ซทรีต้นนี้จากซ้ายไปขวาโดยที่ตัวกระทำตัวแรกที่เราพบคือ เครื่องหมายเท่ากับ (=) จึงนำไปเป็นรากของทรีจากนั้นทางด้านซ้ายของเครื่องหมายเท่ากับก็คือ x เราก็นำไปเป็นลูกทางซ้ายของต้นไม้ จากนั้นทางด้านขวาของเครื่องหมายเท่ากับนั้นคือ y + Z - 2 ซึ่งก็จะต้องเริ่มดูตัวกระทำอีกว่าเป็นตัวอะไรซึ่งก็คือเครื่องหมายบวก (+) จึงนำมาเป็นลูกทางขวาของทรีต้นที่มีรากเป็นเครื่องหมายเท่ากับ แต่เครื่องหมายบวกเองก็จะเป็นรากของทรีต้นใหม่ที่สร้างขึ้นซึ่งก็จะมีลูกด้านซ้ายเป็น y และมีลูกทางด้านขวาเป็น Z - 2 ซึ่งก็จะต้องสร้างเป็นทรีต่อไปจนกว่าจะไม่สามารถสร้างต่อได้

จะสังเกตุเห็นได้ว่าจากตัวอย่างข้างต้นนั้น เครื่องหมายบวก และลบมีนัยสำคัญของการกระทำเท่ากันจึงมีการสร้างทรีจากซ้ายไปขวาได้อย่างปกติแต่หากว่าการกระทำต่างๆ นั้นมีนัยสำคัญไม่เท่ากันเราก็จะต้องคำนึงถึงจุดนี้ด้วยดังตัวอย่างต่อไปนี้

fetch input : x = y * Z + 2



รูปที่ 3-3 แสดงพาร์ซทรีของคำสั่งที่มีนัยสำคัญของการกระทำไม่เท่ากัน

จากรูปที่ 3-3 จะเป็นได้ว่าเครื่องหมายคูณ (*) กับเครื่องหมายบวกนั้น จะมีนัยสำคัญของการกระทำไม่เท่ากันโดยที่เครื่องหมายคูณนั้นจะมีนัยสำคัญมากกว่าเครื่องหมายบวกฉะนั้นทรีที่สร้างขึ้นนั้นก็จะต้องเอา y * Z ไว้ลึกสุด หรือหมายถึงจะต้องมีการประมวลผลก่อน จากนั้นจึงทำการนำค่าที่ได้มาบวกด้วย 2 นั่นเอง

3.5 การกระทำตามคำสั่ง (Execution)

เมื่อผ่านขั้นตอนของการพาร์ซทรีมาแล้ว เราก็จะได้โครงสร้างของคำสั่งนั้นๆ ซึ่งจะนำมากระทำตามคำสั่งนั้นๆ โดยการกระทำจากทรีโหนดที่ลึกที่สุด (leave node) ของทรีก่อน แล้วจึงคืนค่ากลับ (return) ไปให้กับโหนดที่เป็นราก (root node) เพื่อให้ส่วนของทรีที่อยู่สูงกว่าการะทำตามคำสั่งต่อไปจนถึงจุดสูงสุดของทรีนั่นเอง

โดยหากเราลองมาพิจารณาจากรูปที่ 3-1 ก็จะได้การกระทำตามคำสั่งดังต่อไปนี้

1. พิจารณาจากทรีโหนดที่ลึกที่สุด ซึ่งก็คือทรีของ Z - 2 ซึ่งเราก็จะกระทำการลบค่า Z ด้วย 2 แล้วคืนค่าผลลัพธ์กลับขึ้นไปยังตำแหน่งรากของทรี สมมติให้ค่าของ Z เท่ากับ 10 เราจะได้ค่า 8 คืนกลับขึ้นไป
2. พิจารณาจากทรีที่ถัดขึ้นไป ซึ่งก็คือทรีของ y + <ผลลัพธ์ของทรีต้นที่ผ่านมา> ซึ่งเราก็จะกระทำการบวกค่า y ด้วย ผลลัพธ์ที่ได้จากทรีต้นที่ผ่านมาซึ่งคือ 8 แล้วคืนค่าผลลัพธ์กลับขึ้นไปยังตำแหน่งรากของทรี สมมติให้ค่าของ y เท่ากับ 10 เราจะได้ค่า 18 คืนกลับขึ้นไป
3. พิจารณาจากทรีที่ถัดขึ้นไปอีก ซึ่งก็คือทรีของ x = <ผลลัพธ์ของทรีต้นที่ผ่านมา> ซึ่งเราก็จะทำการกำหนดค่าให้กับ x ด้วยผลลัพธ์ที่ได้จากทรีต้อนที่ผ่านมาซึ่งคือ 18 แล้วคืนค่าผลลัพธ์กลับขึ้นไปยังตำแหน่งรากของทรี ซึ่งในที่นี้ไม่มีทรีชั้นต่อไปแล้วจึงเป็นอันเสร็จการกระทำตามคำสั่งนี้

บทที่ 4
การออกแบบและการพัฒนาโปรแกรม

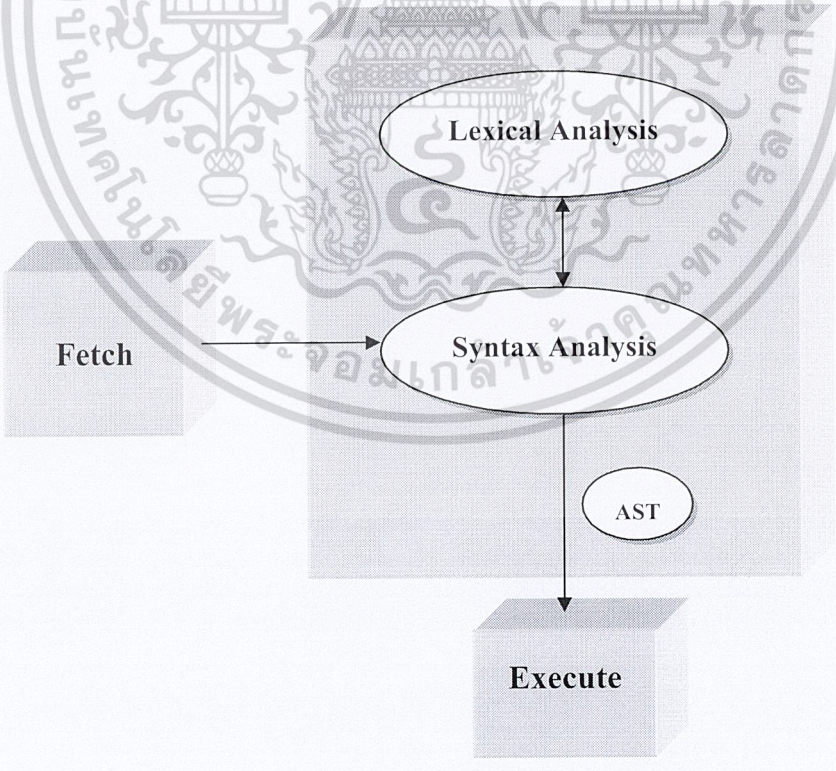
ในการออกแบบและพัฒนาอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กนั้น เราจะสามารถมองโครงสร้างและส่วนประกอบต่างๆ ของโปรแกรมได้อยู่ทั้งหมด 3 ส่วนใหญ่นั่นก็คือ

- ส่วนของการอ่านคำสั่งเข้ามาทีละ 1 ชุดคำสั่ง (Fetch)
- ส่วนของการวิเคราะห์คำสั่ง (Analyze)
- ส่วนของการกระทำตามคำสั่ง (Execution)

โดยที่ส่วนที่มีความยืดหยุ่นและซับซ้อนมากที่สุดจะเป็นในส่วนของการวิเคราะห์คำสั่ง เนื่องจากว่าในการวิเคราะห์คำสั่งนั้นจะมีองค์ประกอบอีกถึงสามส่วนคือ

- ส่วนของการตรวจจับคำ (Lexical Analysis)
- ส่วนตรวจสอบประโยค (Syntax Analysis)
- ส่วนพาร์ซทรี (Parse Tree)

ซึ่งเมื่อเรามองภาพขององค์ประกอบทั้งหมดของโปรแกรมแล้วก็จะได้ออกมาดังรูปนี้



รูปที่ 4-1 แสดงองค์ประกอบของอินเตอร์พรีตเตอร์ไพธอนขนาดเล็ก

4.1 การออกแบบและสร้างส่วนของการอ่านคำสั่ง

ในการอ่านคำสั่งเข้ามาทีละคำสั่งนั้น อันที่จริงแล้วจะต้องสามารถทราบได้ก่อนว่าคำสั่ง และไวยากรณ์ต่างๆ ของภาษาไพธอนส่วนที่จะนำมาใช้นั้นมีอะไรบ้าง โดยการสร้างให้อยู่ในรูปของ เรกกูล่าร์เอ็กซ์เพรสชั่น และบีเอ็นเอฟขึ้นมาทั้งหมดเท่าที่ต้องการใช้ จากนั้นก็จะทำการสร้างส่วนที่รับข้อมูลทั้งหมดเข้ามายังส่วนของการอ่านคำสั่ง แล้วอ่านอักขระออกมาทีละตัว เข้าไปเรื่อยๆ จนส่วนของการตรวจสอบประโยคตรวจพบว่าสิ้นสุดคำสั่งแล้วก็จะนำไปกระทำการสร้างทรีต่อไป

สรุปว่าอันที่จริงแล้วการสร้างส่วนของการอ่านคำสั่งนั้นไม่มีส่วนสำคัญอะไรมากเท่าไหร่เพียงแค่อ่านอักขระเข้ามาที่ละตัวเรื่อยๆ จนหมดข้อมูลเท่านั้นเอง

4.2 การออกแบบและสร้างส่วนของการตรวจจับคำ และการตรวจสอบประโยค

ส่วนของการตรวจจับคำ และส่วนของการตรวจสอบประโยคนั้น อันที่จริงแล้วจะไม่สามารถแยกกันทำงานได้เลย เนื่องจากว่าจะทราบว่าประโยคถูกต้องหรือไม่ ก็จำเป็นต้องทราบว่าแต่ละคำในประโยคนั้นถูกต้องหรือไม่เสียก่อน อาจกล่าวได้ว่าส่วนของการตรวจสอบประโยคนั้นจะไปถามส่วนการตรวจจับคำอีกทีว่าคำที่มีอยู่ในประโยคนั้นถูกต้องหรือไม่ และเมื่อถูกต้องทั้งหมดแล้วส่วนการตรวจสอบประโยคจึงจะทำการตรวจสอบว่าประโยคนั้นถูกต้องหรือไม่โดยการออกแบบนั้นจะทำควบคู่กันไปทั้งสองส่วน

4.2.1 การออกแบบ เรกกูล่าร์เอ็กซ์เพรสชั่น

การออกแบบเรกูล่าร์เอ็กซ์เพรสชั่นนั้น เราจะทำการแยกออกจากกันได้เป็นสองส่วนใหญ่ๆ คือ

- คำที่มีลักษณะคงที่
- คำที่มีลักษณะที่สามารถเปลี่ยนแปลงได้

4.2.1.1 คำที่มีลักษณะคงที่

คำที่มีลักษณะคงที่นั้น ก็จะหมายถึงกลุ่มของตัวอักษรในภาษาไพธอนที่เป็นคำสงวนต่างๆ รวมถึงเครื่องหมายการกระทำต่างๆ ซึ่งเราสามารถที่จะแทนเรกกูล่าร์เอ็กซ์เพรสชั่นด้วยอักขระของคำนั้นๆ ได้เลย เช่น

คำว่า “if” เราสามารถเขียนเรกกูล่าร์เอ็กซ์เพรสชั่นได้ดังนี้

IF “if”

ซึ่งหมายความว่าเรกกูล่าร์เอ็กซ์เพรสชั่นชื่อว่า IF ใช้แทนอักษร i และ f ติดกันเท่านั้น

หรืออย่างเช่น

เครื่องหมายบวก “+” เราสามารถเขียนเรกกูล่าร์เอ็กซ์เพรสชั่นได้ดังนี้

ADD “+”

ซึ่งหมายความว่าเรกกูล่าร์เอ็กซ์เพรสชั่นชื่อว่า ADD ใช้แทนอักษร “+” นั่นเอง

ซึ่งเมื่อลองทำการสร้างเรกกูล่าร์เอ็กซ์เพรสชั่นแบบที่มีลักษณะคงที่ในภาษาไพธอนเท่าที่จำเป็นต้องใช้ในการทำอินเตอร์พรีตเตอร์ขนาดเล็กนี้ก็จะได้ออกมาทั้งหมดเป็นดังนี้

ชื่อ	เรกกูล่าร์เอ็กซ์เพรสชั่น
EQU	“==”
GTE	“>=”
LTE	“<=”
NE1	“<>”
NE2	“!=”
<	“<”
>	“>”
OR	“or”
AND	“and”
ADDE	“+=”
SUBE	“-=”
MULE	“*=”
DIVE	“/=”
POWE	“**=”
MODE	“%=”
POW	“**”
=	“=”
+	“+”
-	“-”
~	“~”
*	“*”
/	“/”
%	“%”
;	“;”
:	“:”
(	“(”
)	“)”
[	“[”
]	“]”
,	“,”

IS	“is”
IN	“in”
NOT	“not”
IF	“if”
ELSE	“else”
ELIF	“elif”
WHILE	“while”
FOR	“for”
BREAK	“break”
CONTINUE	“continue”
PASS	“pass”
RETURN	“return”
PRINT	“print”
DEL	“del”
RANGE	“range”
LEN	“len”
URL	“openURL”

ตารางที่ 4-1 ตารางเรกกูล่าร์เอ็กซ์เพรสชั่นที่มีลักษณะคงที่ที่ได้ทำการออกแบบ

4.2.1.1 คำที่มีลักษณะสามารถเปลี่ยนแปลงได้

คำที่มีลักษณะที่สามารถเปลี่ยนแปลงได้นั้นก็จะหมายถึงกลุ่มของตัวอักษรที่เป็นค่า (value) ซึ่งไม่จำเป็นว่าจะต้องมีค่าเท่านั้นเท่านี้ตายตัวแต่ว่าค่าต่างๆ ทุกค่านั้นจะมีรูปแบบ (format) ของค่าชนิดต่างๆ อยู่ซึ่งแตกต่างกันออกไป ฉะนั้นเราจะต้องศึกษาหาความสัมพันธ์ภายในค่านั้นๆ ว่าจะมีลักษณะเป็นอย่างไร ตัวอย่างเช่น

ตัวเลขตัวหนึ่งตัวใดๆ ก็จะสามารถแทนด้วยเรกกูล่าร์เอ็กซ์เพรสชั่นได้ดังนี้

digit [0-9]

ซึ่งมีความหมายว่าเรกกูล่าร์เอ็กซ์เพรสชั่นชื่อว่า digit ใช้แทนตัวเลข 0, 1, 2, 3, 4, 5, 6, 7, 8 หรือ 9 ตัวใดตัวหนึ่งก็ได้

หรืออย่างเช่น เลขฐานแปด ก็จะสามารถแทนด้วยเรกกูล่าร์เอ็กซ์เพรสชั่นได้ดังนี้

octdigit [0-7]

octinteger 0{octdigit}+

โดยการที่เราจะเขียนเรกกูล่าร์เอ็กซ์เพรสชั่นเพื่อแทนความหมายของอะไรก็ตามที่เริ่มจะซับซ้อนขึ้นเราก็จำเป็นที่จะต้องสร้างเรกกูล่าร์เอ็กซ์เพรสชั่นขึ้นมามากกว่าหนึ่งตัวมาสื่อความหมายร่วมกัน โดยจากตัวอย่างข้างต้นจะประกอบไปด้วยเรกกูล่าร์เอ็กซ์เพรสชั่นทั้งหมดสองตัวคือ

octdigit [0-7]

ซึ่งหมายถึงตัวเลข 0, 1, 2, 3, 4, 5, 6 หรือ 7

และ

octinteger 0{octdigit}+

ซึ่งหมายถึงตัวเลขที่ขึ้นต้นด้วยเลขศูนย์ “0” แล้วตามด้วยเลขอะไรก็ได้ตั้งแต่ 0 ถึง 7 อย่างน้อยหนึ่งตัว

ซึ่งเมื่อลองทำการสร้างเรกกูล่าร์เอ็กซ์เพรสชั่นแบบที่มีลักษณะสามารถเปลี่ยนแปลงได้ในภาษาไพธอนเท่าที่จำเป็นต้องใช้ในการทำอินเตอร์พรีตเตอร์ขนาดเล็กนี้ก็จะได้ออกมาทั้งหมดเป็นดังนี้

ชื่อ	เร็กกูล่าร์เอ็กซ์เพรสชั่น
ientifier	({letter}|"_")({letter}|{digit}|"_")*
letter	({lowercase}|{uppercase})
lowercase	[a-z]
uppercase	[A-Z]
digit	[0-9]
integer	({decimalinteger}|{octinteger}|{hexinteger})
longinteger	{integer}[lL]
intpart	{digit}+
fraction	"."{digit}+
exponent	[eE][+-]?{digit}+
string	'{stringitem1}*'|"{stringitem2}*\"
stringitem1	({stringchar1}|{escapeseq})
stringitem2	({stringchar2}|{escapeseq})
stringchar1	[^\'\n]
stringchar2	[^\"\n]
escapeseq	"\"[\40-\176]
decimalinteger	{nonzerodigit}{digit}*|0
octinteger	0{octdigit}+

hexinteger	0[xX]{hexdigit}+
nonzerodigit	[1-9]
octdigit	[0-7]
hexdigit	({digit}|[a-f]|[A-F])

ตารางที่ 4-2 ตารางเรกกูล่าร์เอ็กซ์เพรสชั่นที่สามารถเปลี่ยนแปลงได้ที่ได้ทำการออกแบบ

โดยเมื่อเราได้ทำการกำหนดเรกกูล่าร์เอ็กซ์เพรสชั่นทั้งหมดของคำที่เราต้องการใช้ในอินเตอร์พรีตเตอร์ขนาดเล็กแล้วเราก็จะทำการออกแบบส่วนที่เกี่ยวข้องกันซึ่งก็คือส่วนของการตรวจสอบประโยค

4.2.2 การออกแบบ บีเอ็นเอฟ

บีเอ็นเอฟนั้นคือการสร้างประโยคของคำสั่งหรือการกระทำต่างๆ ที่จำเป็นต้องใช้ในอินเตอร์พรีตเตอร์ โดยจะต้องเริ่มสร้างจากส่วนของประโยคก่อน ซึ่งไม่มีส่วนเกี่ยวข้องกับประโยคอื่น หรือเรียกได้ว่าเป็นชิ้นส่วนหนึ่งของประโยคที่ย่อยที่สุด และไม่มีส่วนอื่นที่ซ้อนเข้าไปอีกแล้ว เช่น

literal → INTEGER | STRING | LONGINTEGER

ซึ่งจากตัวอย่างข้างต้นจะหมายถึง บีเอ็นเอฟ ที่ชื่อว่า literal นั้นเป็นส่วนหนึ่งของประโยคที่อาจจะนำไปใช้กับประโยคใดๆ ก็ได้ที่มีส่วนประกอบของประโยคเป็นเลข integer, สตริงค์ หรือ long integer

เมื่อเราทำการกำหนดส่วนย่อยของประโยคได้ทั้งหมดแล้ว เราก็จะนำส่วนประกอบต่างๆ เหล่านี้มารวมกันเป็นประโยคที่ใหญ่ขึ้น ซึ่งเมื่อรวมแล้วก็ตามทีก็ยังอาจจะนำประโยคที่ได้จากการรวมของส่วนย่อย ไปสร้างเป็นประโยคที่ใหญ่ขึ้นและมีความหมายที่ซับซ้อนขึ้นต่อไป เช่น

augmented_stmt → target augop expression_list

จากตัวอย่างจะหมายถึง บีเอ็นเอฟ ที่ชื่อว่า augmented_stmt นั้นประกอบไปด้วย บีเอ็นเอฟ อื่นๆ ที่เป็นส่วนย่อยของประโยคที่ชื่อว่า target, augop และ expression_list ซึ่งในส่วนย่อยๆ ของ augmented_stmt เองก็อาจจะประกอบไปด้วยส่วนย่อยเข้าไปอีกก็เป็นได้เช่น บีเอ็นเอฟ ที่ชื่อว่า target ก็จะมีลักษณะเป็นดังต่อไปนี้

target → IDENTIFIER | subscription

จะเห็นได้ว่า target นั้นจะประกอบไปด้วย identifier หรืออาจจะเป็นส่วนย่อยของประโยคที่ชื่อว่า subscription ก็ได้ซึ่ง subscription ก็จะประกอบไปด้วยประโยคย่อยอื่นต่อๆ ไปอย่างนี้เรื่อยๆ จนกว่าจะถึงส่วนย่อยที่สุดที่ไม่ได้ประกอบไปด้วยส่วนของประโยคใดๆ อีกแล้ว

ซึ่งเมื่อลองทำการสร้างบีเอ็นเอฟในภาษาไพธอนเท่าที่จำเป็นต้องใช้ในการทำอินเตอร์พรีตเตอร์ขนาดเล็กขึ้นมาแล้วก็จะพบว่าจะได้ บีเอ็นเอฟ ออกมาเป็นจำนวนมาก จึงจะได้ขอยกตัวอย่างมาเพียงบางตัวดังนี้

ชื่อ	บีเอ็นเอฟ
Pimary	atom | target
atom	literal | enclosure
target	IDENTIFIER | subscription
literal	INTEGER | STRING | LONGINTEGER
subscription	primary '[' expression_list ']'
enclosure	'(' expression_list_ ')' | '[' expression_list_ ']'
compound_stmt	if_stmt | while_stmt | for_stmt
if_stmt	IF expression ':' suite elif_stmt else_stmt
elif_stmt	ELIF expression ':' suite elif_stmt
else_stmt	ELSE ':' suite | <empty string>
while_stmt	WHILE expression ':' suite else_stmt
for_stmt	FOR target IN expression_list ':' suite else_stmt
range_stmt	RANGE '(' expression ')'
len_stmt	LEN '(' expression_list ')'
expression_stmt	expression_list
expression_list	expression expression_
expression_	',' expression expression_ | ',' | <empty string>
expression	or_test
or_test	and_test | or_test OR and_test
and_test	not_test | and_test AND not_test
not_test	comparison | NOT not_test

ตารางที่ 4-3 ตารางบีเอ็นเอฟบางส่วนของภาษาไพธอนที่ได้ทำการออกแบบ

ทั้งเรกกูล่าร์เอ็กซ์เพรสชั่น และบีเอ็นเอฟ ที่เราได้ทำการสร้างขึ้นทั้งหมดนั้นจะมีส่วนเกี่ยวข้องกันอย่างมาก เนื่องจากหาก เรกกูล่าร์เอ็กซ์เพรสชั่นตัวใดมีการเปลี่ยนแปลงเกิดขึ้น ย่อมส่งผลทำให้ส่วนของประโยคที่มีการใช้ เรกกูล่าร์เอ็กซ์เพรสชั่นตัวนั้น เปลี่ยนตามไปด้วย ฉะนั้นในบางครั้งในช่วงการ

ออกแบบอยู่นั้น เมื่อได้ข้อสรุปออกมา แล้วนำไปทำการเขียนโปรแกรม ก็อาจพบผลที่ไม่เป็นไปตามที่ต้องการซึ่งก็จะต้องกลับมาแก้ไขทั้งเรกกูล่าร์เอ็กซ์เพรสชั่นและ บีเอ็นเอฟด้วย จึงทำให้โครงสร้างของภาษาเป็นสิ่งที่สำคัญมาก คือต้องทำการสร้างให้ดี เพราะการแก้ไขแต่ละครั้งนั้นจะส่งผลกระทบกับส่วนต่างๆ เป็นอย่างมาก

4.2.3 การออกแบบเรกกูล่าร์เอ็กซ์เพรสชั่นและบีเอ็นเอฟ สำหรับทำงานร่วมกับบราวเซอร์ที่โปรแกรมได้

เนื่องจากการพัฒนาและจัดสร้างอินเตอร์พรีตเตอร์ในการวิจัยครั้งนี้มีจุดประสงค์ที่จะทำมาเพื่อนำไปใช้กับบราวเซอร์ที่พัฒนาขึ้นให้สามารถโปรแกรมได้ ฉะนั้นด้วยตัวภาษาไพธอนเองนั้นจะไม่มีชุดคำสั่งที่จะสามารถสั่งบราวเซอร์ให้ทำงานได้ดังนั้นจึงต้องมีการเพิ่มเติมคำสั่งการทำงานเพื่อให้เป็นประโยชน์ในการทำงานร่วมกับบราวเซอร์

แต่ด้วยความจำกัดของภาษาจาวาบนโทรศัพท์มือถือ (J2ME : java 2 micro edition) ที่ใช้สร้างตัวบราวเซอร์ขึ้นมานั้น ยังมีความสามารถไม่มากพอที่จะทำงานหลายๆ อย่างได้อย่างเต็มที่ เช่น ส่ง short message, ส่งอีเมลล์ (e-mail) หรือติดต่อกับพอร์ตอินฟาเรด เป็นต้น ซึ่งเป็นผลทำให้ไม่สามารถสร้างคำสั่งที่เกี่ยวข้องกับจุดต่างๆ ดังที่กล่าวมาแล้วได้ ทำให้ในการพัฒนาครั้งนี้นั้นได้ออกแบบคำสั่งที่ใช้งานร่วมกับบราวเซอร์ไว้เพียงคำสั่งเดียวก็คือคำสั่งในการสั่งให้บราวเซอร์เปิดเว็ปไซต์ (web site) หรือเว็ปเพจ (web page)

โดยที่คำสั่งที่ใช้สำหรับเปิดเว็ปไซต์นั้นจะสามารถที่จะกำหนดเวลาในการเปิดได้ ฉะนั้นก็จะมีการเพิ่มเรกกูล่าร์เอ็กซ์เพรสชั่น ที่ใช้อ้างถึงชนิดข้อมูลชนิดใหม่ เรียกว่าไทม์ (time) ขึ้น

4.2.3.1 ชนิดข้อมูลชนิดไทม์

ข้อมูลชนิดไทม์นั้นจะแบ่งออกได้เป็นสองลักษณะคือ

- Time Point หมายถึงเป็นการระบุเวลาขณะใดขณะหนึ่ง เช่น [17:08:34|15/03/2003]
- Time Interval หมายถึงการระบุจำนวนของเวลา เช่น 2day ซึ่งหมายถึงเป็นจำนวนเวลา 2 วัน

จากลักษณะของข้อมูลชนิดไทม์ดังที่ได้กล่าวมาแล้ว จึงสามารถออกแบบเรกกูล่าร์เอ็กซ์เพรสชั่นสำหรับลักษณะของข้อมูลชนิดไทม์ได้เป็นดังนี้

ชื่อ	เร็กกูล่าร์เอ็กซ์เพรสชั่น
year_	[0]*{nonzerodigit}{digit}*
date29	([0]?{nonzerodigit}|[1-2]{digit})
month29	{date29}"/"[0]?"2/"{year_}
month31	({date29}|"3"[0-1])"/"([0]?[13578]|"10"|"12")"/"{year_}
month30	({date29}|"30")"/"([0]?[469]|"11")"/"{year_}

date	({month29}|{month30}|{month31})
time_	([0-1]?{digit}|"2"[0-3]):"[0-5]?{digit}":"[0-5]?{digit}
time	"["{time_}"|"{date}"]"
second	{integer}"sec"
minute	{integer}"min"
hour	{integer}"hr"
day	{integer}"day"
month	{integer}"mt"
year	{integer}"yr"

ตารางที่ 4-4 ตารางเรกกูล่าร์เอ็กซ์เพรสชั่นสำหรับข้อมูลชนิดไทม์

4.2.3.2 การกระทำกับข้อมูลชนิดไทม์

เมื่อมีชนิดของข้อมูลก็ย่อมจะต้องมีการกระทำของข้อมูลชนิดนั้นๆ ด้วยซึ่งการกระทำของข้อมูลชนิดไทม์นั้นจะมีดังนี้

การกระทำกับข้อมูลชนิดไทม์		
Time Point	=	Time Point + Time Interval
Time Point	=	Time Point - Time Interval
Time Interval	=	Time Interval + Time Interval
Time Interval	=	Time Interval - Time Interval
Time Interval	=	Time Point – Time Point
Time Interval	=	Time Interval * Number
Number	=	Time Interval / Time Interval
Time Interval	=	Time Interval / Number
Time Interval	=	Time Interval ** Number
Number	=	Time Interval % Time Interval
Time Interval	=	Time Interval % Number

ตารางที่ 4-5 ตารางการกระทำลักษณะต่างๆ สำหรับข้อมูลชินดไทม์

4.2.3.3 คำสั่งในการเปิดเว็ปไซด์

คำสั่งในการเปิดเว็ปไซด์เราจะต้องทำการออกแบบ บีเอ็นเอฟ ของคำสั่งนี้ซึ่งให้ชื่อของคำสั่งนี้ว่า openURL และมีรูปแบบการทำงานสองลักษณะดังต่อไปนี้

- เปิดเว็ปไซด์ทันที openURL(<url>)
- เปิดเว็ปไซด์ตามเวลาที่กำหนด openURL(<url>, <time>)

จากลักษณะของคำสั่งข้างต้นที่ได้กล่าวมาแล้ว จึงสามารถออกแบบ บีเอ็นเอฟ ตามลักษณะของคำสั่งการเปิดเว็ปไซด์ได้เป็นดังนี้

ชื่อ	บีเอ็นเอฟ
url_stmt	URL '(' STRING ')' | URL '(' STRING ',' TIME ')'

ตารางที่ 4-6 ตารางบีเอ็นเอฟของคำสั่งการเปิดเว็ปไซด์

4.2.3.3 คำสั่งในการส่งเมลล์

คำสั่งในการส่งเมลล์เราจะต้องทำการออกแบบ บีเอ็นเอฟ ของคำสั่งนี้ซึ่งให้ชื่อของคำสั่งนี้ว่า sendMail และมีรูปแบบการทำงานสองลักษณะดังต่อไปนี้

- ส่งเมลล์ทันที sendMail(<String>, <String>, <String>, <String>)
- ส่งเมลล์ตามเวลาที่กำหนด openURL(<String>, <String>, <String>, <String>, <time>)

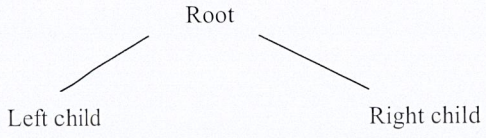
จากลักษณะของคำสั่งข้างต้นที่ได้กล่าวมาแล้ว จึงสามารถออกแบบ บีเอ็นเอฟ ตามลักษณะของคำสั่งการเปิดเว็ปไซด์ได้เป็นดังนี้

ชื่อ	บีเอ็นเอฟ
mail_stmt	MAIL '(' STRING ',' STRING ',' STRING ',' STRING ')' | MAIL '(' STRING ',' STRING ',' STRING ',' STRING ',' TIME ')'

ตารางที่ 4-7 ตารางบีเอ็นเอฟของคำสั่งการส่งเมลล์

4.2.4 การออกแบบและสร้างพาร์ซทรี

การสร้างพาร์ซทรีนั้น จะเป็นการนำคำสั่งที่ผ่านการตรวจสอบประโยคว่าถูกต้องแล้วเท่านั้นถึงจะนำมาสร้างพาร์ซทรี โดยในการออกแบบนี้จะมองทรีออกเป็นสองลักษณะคือ ทรีที่มี 2 กิ่ง และ ทรีที่มี 3 กิ่ง



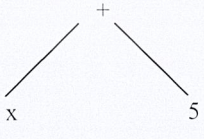
รูปที่ 4-2 แสดงพาร์ซทรีแบบสองกิ่ง

รูปที่ 4-3 แสดงพาร์ซทรีแบบสามกิ่ง

1. พาร์ซทรีแบบสองกิ่ง

สำหรับการออกแบบพาร์ซทรีแบบสองกิ่งนั้นก็จะสร้างไว้สำหรับใช้กับคำสั่งโดยทั่วไปที่กระทำตามตัวกระทำโดยตรง และไม่มีเงื่อนไขในการกระทำ หากเจอคำสั่งลักษณะเช่นนี้ เราก็จะนำเอาตัวกระทำเป็นรากของทรีส่วนที่เหลือก็จะเป็นลูกทางซ้ายและลูกทางขวาตามลำดับ เช่น

x + 5 จะได้พาร์ซทรีดังนี้



รูปที่ 4-4 แสดงตัวอย่างการสร้างทรีแบบสองกิ่งจากคำสั่งไพธอน x + 5

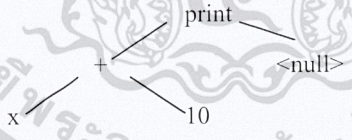
จากตัวอย่างจะเห็นได้ว่าการกระทำของคำสั่งคือเครื่องหมายบวก ซึ่งเราก็จะนำมาเป็นรากของทรี และทางด้านซ้ายของการกระทำก็คือ x เราก็จะนำมาเป็นลูกทางด้านซ้ายของทรี และในทำนองเดียวกัน 5 อยู่ทางด้านขวาของการกระทำเราก็นำมาเป็นลูกทางขวาของทรี

z = x * 5 จะได้พาร์ซทรีดังนี้

*รูปที่ 4-5 แสดงตัวอย่างการสร้างทรีแบบสองกิ่งจากคำสั่งไพธอน z = x * 5*

จากตัวอย่างจะเห็นได้ว่าการกระทำของคำสั่งคือการกำหนดค่า ซึ่งเราก็จะนำมาเป็นรากของทรี และทางด้านซ้ายของการกระทำก็คือ z เราก็จะนำมาเป็นลูกทางด้านซ้ายของทรี และในทำนองเดียวกัน x * 5 อยู่ทางด้านขวาของการกระทำเราก็นำมาเป็นลูกทางขวาของทรี แต่ว่าลูกทางด้านขวานั้นมีลักษณะเป็นทรีอีก ฉะนั้นเราก็ต้องมาทำการสร้างต่อโดยการกระทำของทรี x * 5 นั้นก็คือการคูณ เราก็จะนำมาเป็นรากของทรีและทางด้านซ้ายของการกระทำก็คือ x เราก็จะนำมาเป็นลูกทางด้านซ้ายของทรี และในทำนองเดียวกัน 5 อยู่ทางด้านขวาของการกระทำเราก็นำมาเป็นลูกทางขวาของทรี

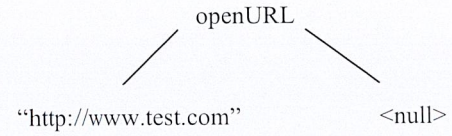
print x + 10 จะได้พาร์ซทรีดังนี้



รูปที่ 4-6 แสดงตัวอย่างการสร้างทรีแบบสองกิ่งจากคำสั่งไพธอน print x + 10

จากตัวอย่างจะเห็นได้ว่าการกระทำของคำสั่งคือการแสดงผล (print) ซึ่งเราก็จะนำมาเป็นรากของทรี โดยที่คำสั่งการแสดงผลนั้นจะไม่มีลูกทางด้านขวา เพราะเป็นแค่การแสดงผลออกมาโดยที่ไม่ได้กระทำกับค่าใด จึงกำหนดให้ลูกทางด้านขวาเป็นค่า null ส่วนทางด้านซ้ายเป็นทรีของการบวก เราก็จะนำการบวกเป็นราก ส่วนลูกทางด้านซ้ายก็คือ z และ ลูกทางด้านขวาเป็น 5

openURL(“http://www.test.com”) จะได้พาร์ซทรีดังนี้



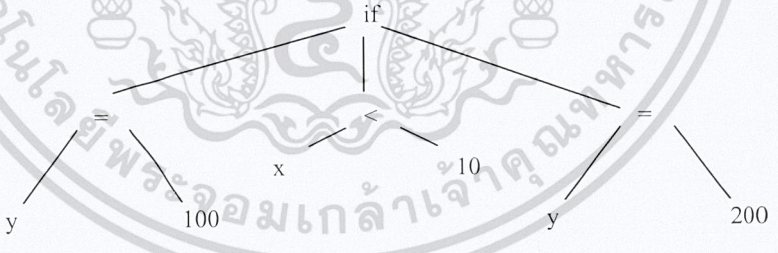
รูปที่ 4-7 แสดงตัวอย่างการสร้างทรีแบบสองกิ่งจากคำสั่งการเปิดเว็ปไซต์

จากตัวอย่างคำสั่งการเปิดเว็ปไซต์นั้น จะมีลักษณะเหมือนกับคำสั่งการแสดงผลทางหน้าจอที่ได้กล่าวไปแล้ว คือจะไม่มีลูกทางด้านขวาเนื่องจากไม่มีการกระทำกับค่าใด โดยจะมีเพียงลูกทางด้านซ้ายซึ่งก็คือยูอาร์เอล ที่ต้องการจะทำการเปิดนั่นเอง

2. พาร์ซทรีแบบสามกิ่ง

สำหรับการออกแบบพาร์ซทรีแบบสามกิ่งนั้นก็จะสร้างไว้สำหรับใช้กับคำสั่งที่มีเงื่อนไขในการกระทำหรือไม่กระทำซึ่งทำให้มีผลของการกระทำมากว่า 1 การกระทำหากเจอคำสั่งลักษณะเช่นนี้ เราก็จะนำเอาชื่อคำสั่งซึ่งจะบ่งบอกถึงการกระทำในภายหลัง มาเป็นรากของทรีส่วนลูกทางด้านซ้ายนั้นจะให้เป็นทรีของคำสั่งที่จะถูกกระทำเมื่อเงื่อนไขเป็นจริงส่วนลูกทางด้านขวาจะเป็นทรีของคำสั่งที่จะถูกกระทำเมื่อเงื่อนไขเป็นเท็จ และสุดท้ายคือนำเงื่อนไขของคำสั่งมาเป็นลูกตรงกลางของทรี เช่น

if x < 10 ; y = 100
else : y = 200 จะได้พาร์ซทรีดังนี้



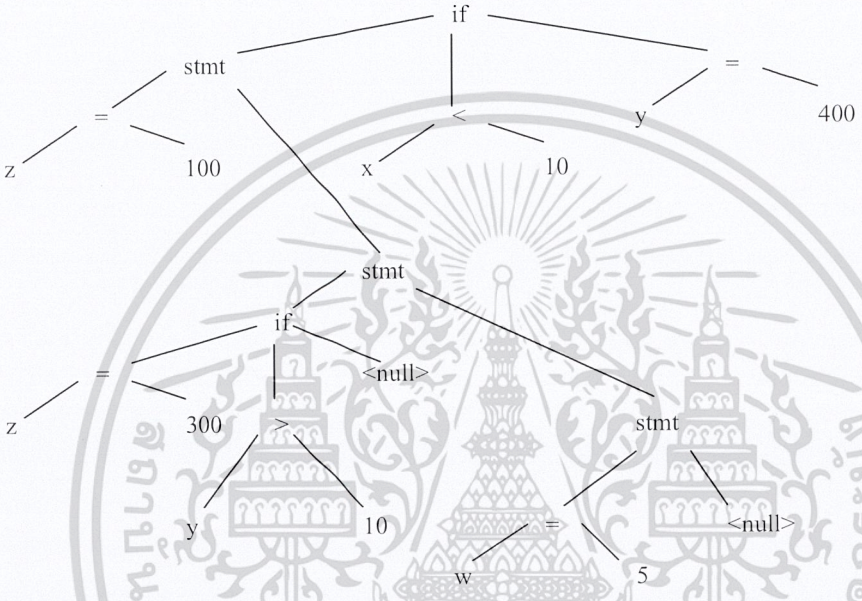
รูปที่ 4-8 แสดงตัวอย่างการสร้างทรีแบบสามกิ่งจากคำสั่งไพธอน if x < 10 : y = 10 else : y = 200

จากตัวอย่างจะเห็นได้ว่าชื่อของคำสั่งนั้นก็คือ "if" ซึ่งเราก็จะนำมาเป็นรากของทรี และทางด้านซ้ายของทรีนั้น ก็จะเป็นทรีของคำสั่งที่จะกระทำเมื่อเงื่อนไขเป็นจริง ซึ่งนั่นก็คือ y = 100 ส่วนลูกทางด้านขวานั้นก็จะเป็นทรีของคำสั่งที่จะกระทำเมื่อเงื่อนไขเป็นเท็จ ซึ่งนั้นก็คือ y = 200 ส่วนสุดท้ายคือลูกตรงกลางนั้นก็จะเป็นเงื่อนไขของคำสั่งซึ่งก็คือ x < 10

โดยเราจะสังเกตุได้ว่าลูกทั้งสามกิ่งนั้นจะเป็นทรีของแต่ละคำสั่งอีกทีซึ่งจะมีสองหรือสามกิ่งต่อไปอีกหรือไม่ก็ขึ้นอยู่กับคำสั่งหรือการกระทำนั้นๆ จะเป็นอะไรนั่นเอง

```
if x < 10 :
   z = 100
   if y > 10 :
      z = 300
   w = 5
else : y = 400
```

จะได้พาร์ซทรีดังนี้



รูปที่ 4-9 แสดงตัวอย่างการสร้างทรีแบบสามกิ่งจากคำสั่งไพธอนที่ซ้อนหลายชั้น

จากตัวอย่างนี้จะแสดงให้เห็นถึงการสร้างทรีแบบสามกิ่งซ้อนกันหลายๆ ชั้นและภายใต้คำสั่ง “if” จะมีหลายคำสั่งอยู่ภายในคำสั่ง “if” ซึ่งจะสังเกตเห็นได้ว่าหากมีคำสั่งเพียงคำสั่งเดียวอยู่ภายใต้คำสั่ง “if” กิ่งทางด้านซ้ายของคำสั่งจะเป็นการกระทำของคำสั่งที่อยู่ภายใต้ “if” ทันทีแต่หากมีคำสั่งมากกว่า 1 คำสั่งกิ่งทางด้านซ้ายนั้นจะเป็นทรีที่ชื่อว่า “stmt” ซึ่งจะบรรจุคำสั่งไว้ทางด้านซ้ายของทรีและทางด้านขวาก็จะเป็นคำสั่งถัดไปเรื่อยๆ จะกระทั่งลูกทางด้านขวาของ “stmt” เป็น null ซึ่งแสดงว่าไม่มีคำสั่งต่อจากนี้แล้ว

และจากตัวอย่างให้สังเกตคำสั่ง “if” ที่อยู่ภายใต้คำสั่ง “if” อีกทีนั้น จะไม่มีส่วนของ “else” ซึ่งทำให้ลูกทางด้านขวาของตัวมันเป็น null

4.3 การออกแบบและสร้างส่วนของการกระทำตามคำสั่ง

ส่วนของการกระรทำตามคำสั่งนั้นเป็นส่วนสุดท้ายของกระบวนการการอินเตอร์พรีต ซึ่งถ้าดูเผินๆ แล้วอาจเห็นว่าส่วนของการกระทำตามคำสั่งนั้นก็เพียงแค่กระทำไปตามลักษณะของคำสั่งที่ได้รับมาจากพาร์ซทรี แต่จริงๆ แล้วหน้าที่ของส่วนนี้จะมีอยู่ด้วยกันสามส่วนก็คือ

- ตรวจสอบความผิดพลาดของชนิดตัวแปรที่นำมากระทำตามคำสั่งต่างๆ
- กระทำตามคำสั่งที่รับเข้ามา
- จัดการกับตัวแปรต่างๆ ที่ต้องนำไปกระทำ หรือได้จากการกระทำนั้นๆ

4.3.1 การตรวจสอบความผิดพลาดของชนิดตัวแปรที่นำมากระทำตามคำสั่ง

เนื่องจากการ สร้างเรกกูล่าร์เอ็กซ์เพรสชั่น และการสร้าง บีเอ็นเอฟนั้นจะเป็นการกำหนดความสมพันธ์ระหว่างกลุ่มคำและประโยคของภาษา โดยจัดเป็นกลุ่มๆ เป็นพวกๆ เท่านั้น ไม่สามารถที่จะจัดการได้ถึงขั้นเป็นคำๆ หรือเป็นตัวๆ ได้ หรือหากทำได้ก็จะทำให้มีความซับซ้อนในการทำงานมากจนเกินไปได้

โดยการกระทำดังกล่าวนี้สามารถยกหน้าที่ให้มาทำในส่วนของการกระทำตามคำสั่งได้ เพราะว่าในส่วนนี้นั้น จะเข้าไปกระทำยังข้อมูลที่ลึกที่สุดของแต่ละส่วนของคำสั่ง ซึ่งจะทำให้สามารถทราบได้ว่าการกระทำนั้นสามารถกระทำได้หรือไม่ เพื่อให้เข้าใจมากขึ้นจะยกตัวอย่างให้เห็นลักษณะดังกล่าวดังต่อไปนี้

สมมติเรากำหนดให้

literal → INTEGER | STRING | LONGINTEGER

และ

expression → literal '+' literal

ซึ่งนั่นหมายความว่าการกระทำการบวกนั้นสามารถกระทำได้โดยการนำเอา literal บวกกับ literal แต่ว่าใน literal เองนั้นมีตัวแปรอยู่ด้วยกัน 3 ชนิดคือ integer, string และ long integer หากในการใช้คำสั่งจาก บีเอ็นเอฟ นี้เป็น 10 + 20 ซึ่งก็เป็น integer บวกกับ integer ซึ่งก็จะไม่มีปัญหาอะไร แต่หากลองเขียนใหม่เป็น "test" + 10 อย่างนี้ การตรวจสอบประโยคนั้นก็ไม่สามารถตรวจพบได้ว่ามีความผิดพลาดเกิดขึ้นเนื่องจาก string และ integer ต่างก็ถูกแทนด้วย literal และการบวกก็สามารถกระทำได้โดยการนำเอา literal บวกกับ literal

จึงทำให้เราเห็นได้ว่า จำเป็นอย่างยิ่งที่ส่วนของการกระทำตามคำสั่งนั้นจะต้องทำการตรวจสอบชนิดของข้อมูลก่อนว่าสามารถจะกระทำตามคำสั่งนั้นๆ ได้หรือไม่

4.3.2 การกระทำตามคำสั่งที่รับเข้ามา

หลังจากที่มีการตรวจสอบได้แล้วว่าชนิดของข้อมูลที่จะถูกกระทำนั้นไม่ขัดแย้งกับการกระทำเรา ก็สามารถที่จะกระทำตามคำสั่งนั้นได้ทันที โดยการเช็คว่าเป็นการกระทำใด ก็กระทำไปตามนั้นได้ทันที

4.3.3 จัดการกับตัวแปรต่างๆ ที่ต้องนำไปกระทำ หรือได้จากการกระทำ

เนื่องจากว่าการกระทำต่างๆ ส่วนใหญ่แล้วมักจะไม่ใช่เป็นการกระทำระหว่างค่าคงที่กับค่าคงที่ แต่มักจะเป็นการกระทำระหว่าง ตัวแปรกับค่าคงที่หรือไม่ก็ตัวแปรกับตัวแปรมากกว่า อีกทั้งผลที่ได้จากการกระทำต่างๆ ก็มักจะนำไปเก็บไว้ในตัวแปรด้วย

ดังนั้นส่วนของการกระทำตามคำสั่งนี้ จะต้องมีการจัดการกับตัวแปรเหล่านี้ โดยจะแบ่งการจัดการกับตัวแปรต่างๆ ได้เป็นสองกรณีคือ

- นำค่าที่ได้จากการกระทำไปเก็บไว้ในตัวแปร
- นำค่าที่เก็บไว้ออกมาใช้เพื่อกระทำคำสั่งต่างๆ

โดยการจัดการทั้งสองกรณีนี้ ก็จะต้องมีสิ่งที่ใช้ในการเก็บและนำออก ซึ่งเราเรียกว่าตารางสัญลักษณ์ (Symbol Table) ซึ่งก็จะหมายถึงตารางที่ใช้เก็บตัวแปรและค่าของตัวแปรต่างๆ ที่เกิดขึ้นในขณะทำงานโปรแกรมหนึ่งๆ หรือ สคริปต์หนึ่งๆ นั่นเอง โดยเราจะสร้างขึ้นโดยการใช้ ตารางแฮ็ช (Hash Table) ซึ่งมีคุณสมบัติที่เหมาะกับการเก็บและนำค่าที่เก็บไว้กลับมาใช้รวมถึงการเปลี่ยนแปลงค่าที่เคยมีอยู่เดิมเป็นค่าใหม่

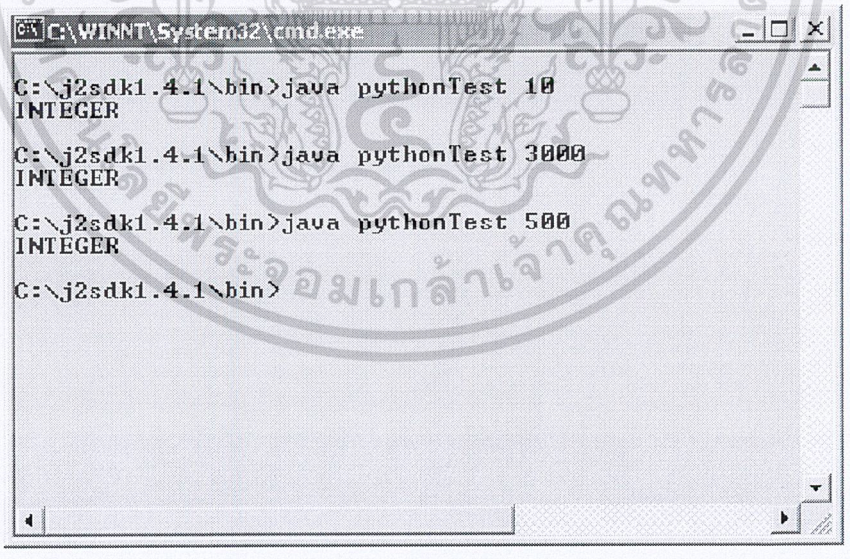
บทที่ 5

ผลการทดสอบ

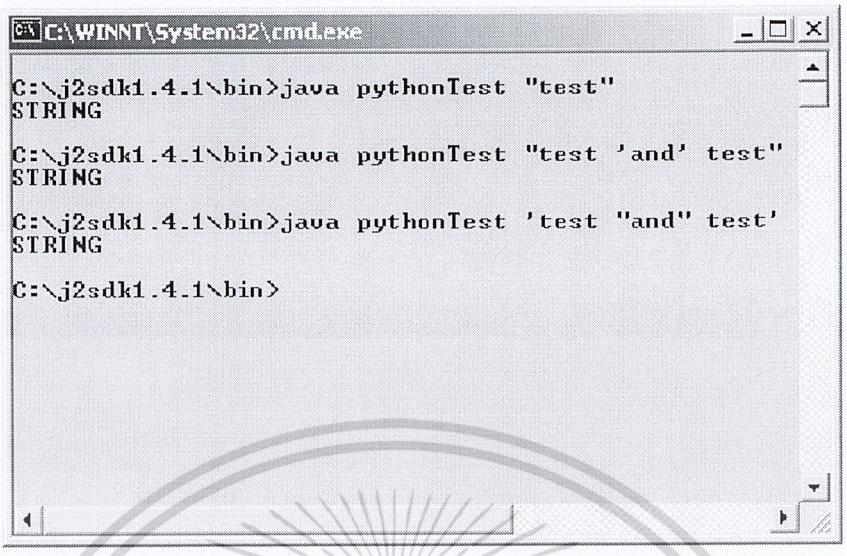
ในการพัฒนาโปรแกรม หรือซอฟต์แวร์นั้นมีขั้นตอนในการทำงานอยู่หลายขั้นตอนซึ่งการทดสอบความถูกต้องของโปรแกรมนั้นก็เป็นหนึ่งในหลายขั้นตอนของการพัฒนาโปรแกรม ซึ่งนับได้ว่ามีความสำคัญมากเพราะจะทำให้เรามั่นใจได้ว่าโปรแกรมที่เราทำการพัฒนาขึ้นมานั้นสามารถนำไปใช้งานได้อย่างถูกต้อง อีกทั้งขั้นตอนการทดสอบความถูกต้องของโปรแกรมนั้นจะทำให้ เราสามารถที่จะค้นพบข้อผิดพลาดบางอย่าง ซึ่งทำให้เราสามารถแก้ไขได้ก่อนที่จะมีการนำไปใช้งานได้จริง

5.1 การทดสอบความถูกต้องของกลุ่มคำที่ได้ทำการกำหนดโดยใช้เรกกูล่าร์เอ็กซ์เพรสชั่น

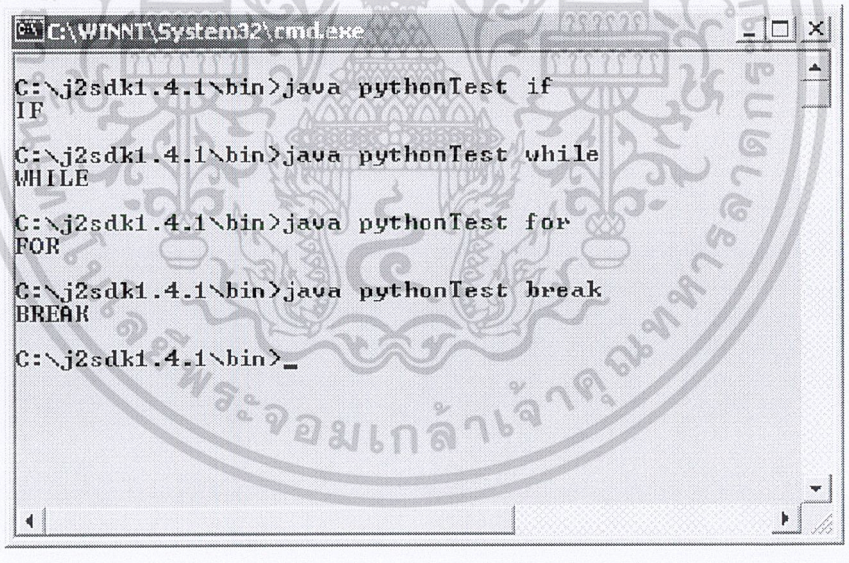
การทดสอบความถูกต้องของกลุ่มคำนั้นเราจะกระทำโดยการนำเฉพาะโค้ดในส่วนของการตรวจจับกลุ่มคำมาทดลองใส่กลุ่มคำทั้งแบบที่ถูก และแบบที่ผิดหลายๆ ตัวอย่าง และทำการแสดงชื่อของเรกกูล่าร์เอ็กซ์เพรสชั่นนั้นๆ ออกมาทางเอาท์พุทถ้าคำที่ทำการทดลองตรงตามเร็กกูล่าร์เอ็กเพสชั่นตัวนั้นๆ และแสดงคำว่า “ERROR” ออกมาทางเอาท์พุทหากคำที่ทดลองไม่ถูกต้องตามเร็กกูล่าร์เอ็กเพสชั่นใดๆ เลย ซึ่งจากการทดลองกับกลุ่มคำต่างๆ จึงได้ผลออกมาดังต่อไปนี้



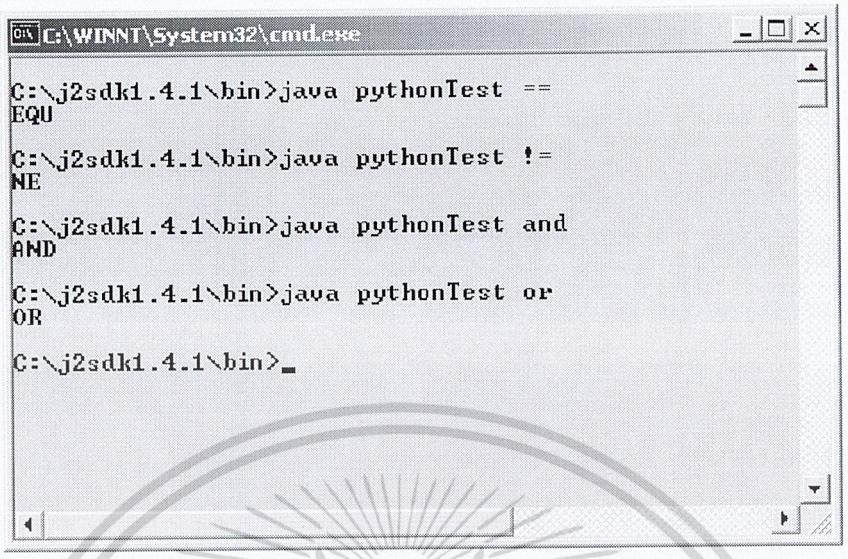
รูปที่ 5-1 แสดงตัวอย่างการทดสอบเรกกูล่าร์เอ็กซ์เพรสชั่นของข้อมูลชนิด integer



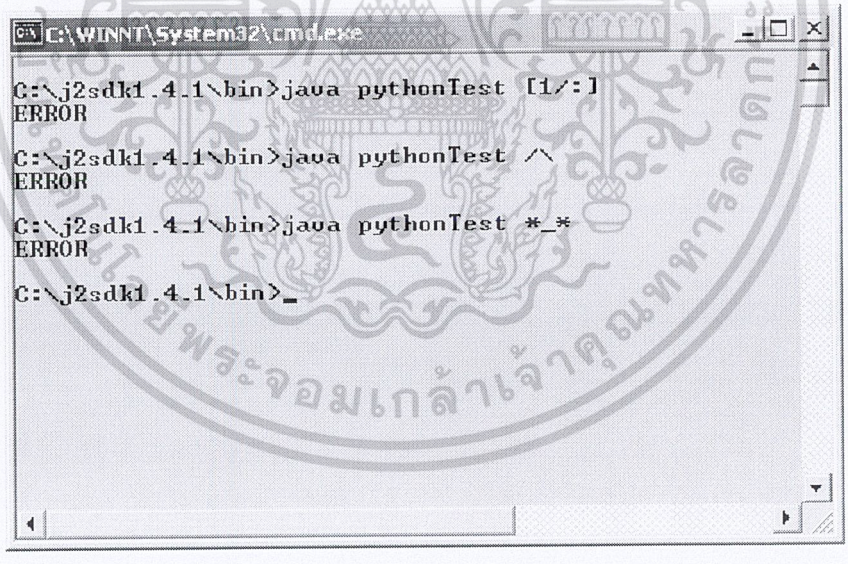
รูปที่ 5-2 แสดงตัวอย่างการทดสอบเรกกูล่าร์เอ็กซ์เพรสชั่นของข้อมูลชนิด string



รูปที่ 5-3 แสดงตัวอย่างการทดสอบเรกกูล่าร์เอ็กซ์เพรสชั่นของคำสงวนต่างๆ



รูปที่ 5-4 แสดงตัวอย่างการทดสอบเรกกูล่าร์เอ็กซ์เพรสชั่นของตัวกระทำแบบต่างๆ

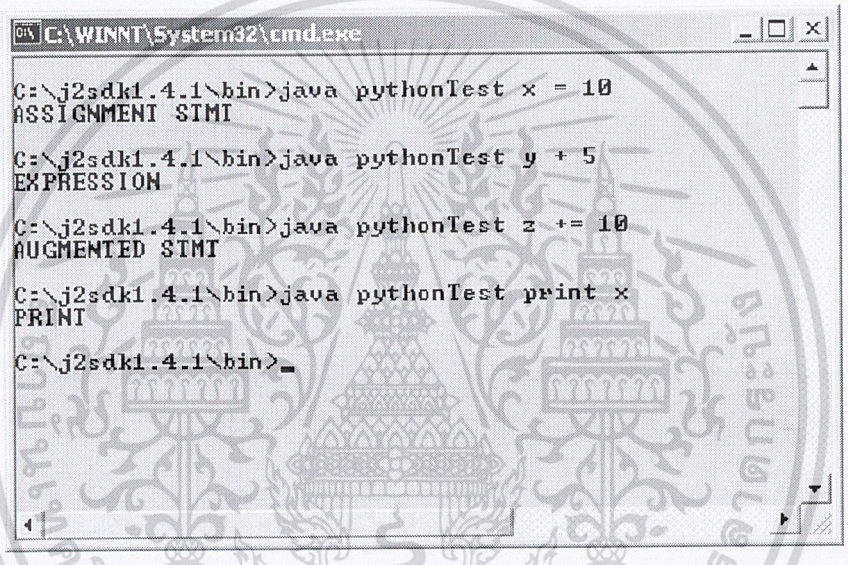


รูปที่ 5-5 แสดงตัวอย่างการทดสอบคำที่ไม่ถูกต้องตามเร็กกูล่าร์เอ็กเพสชั่นใดเลย

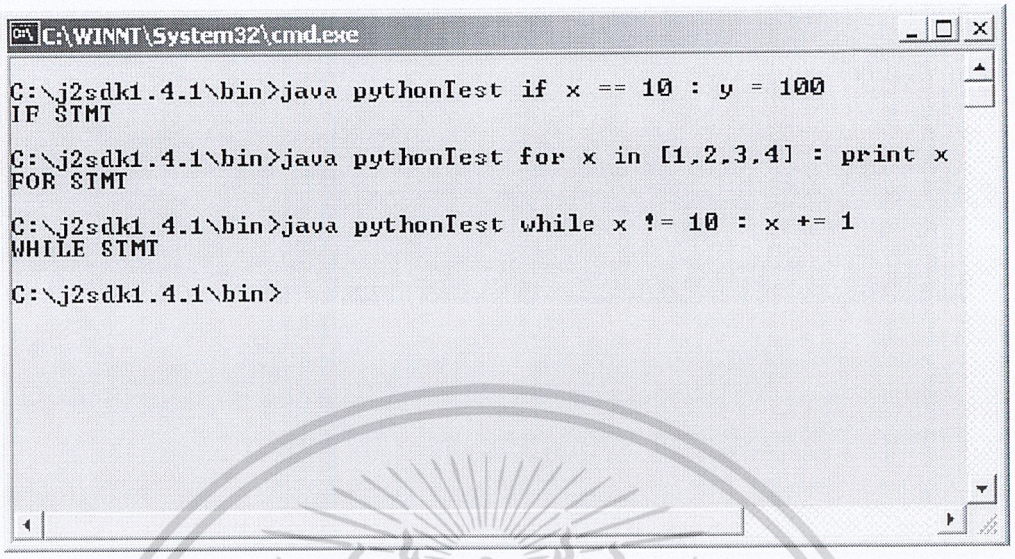
5.2 การทดสอบความถูกต้องของประโยคที่ได้ทำการกำหนดโดยใช้บีเอ็นเอฟ

การทดสอบความถูกต้องของประโยคนั้นเราจะกระทำโดยการนำส่วนของการตรวจจับกลุ่มคำมารวมกับส่วนของการตรวจสอบประโยคที่ทำการสร้างขึ้น โดยจะมีการทดลองใส่ประโยคทั้งแบบที่ถูก และแบบที่ผิดหลายๆ ตัวอย่าง และทำการแสดงชื่อของบีเอ็นเอฟนั้นๆ ออกมาทางเอาท์พุทถ้าประโยคที่ทำการทดลองตรงตามบีเอ็นเอฟตัวนั้นๆ และแสดงคำว่า “ERROR” ออกมาทางเอาท์พุทหากประโยคที่ทดลองไม่ถูกต้องตามบีเอ็นเอฟใดๆ เลย

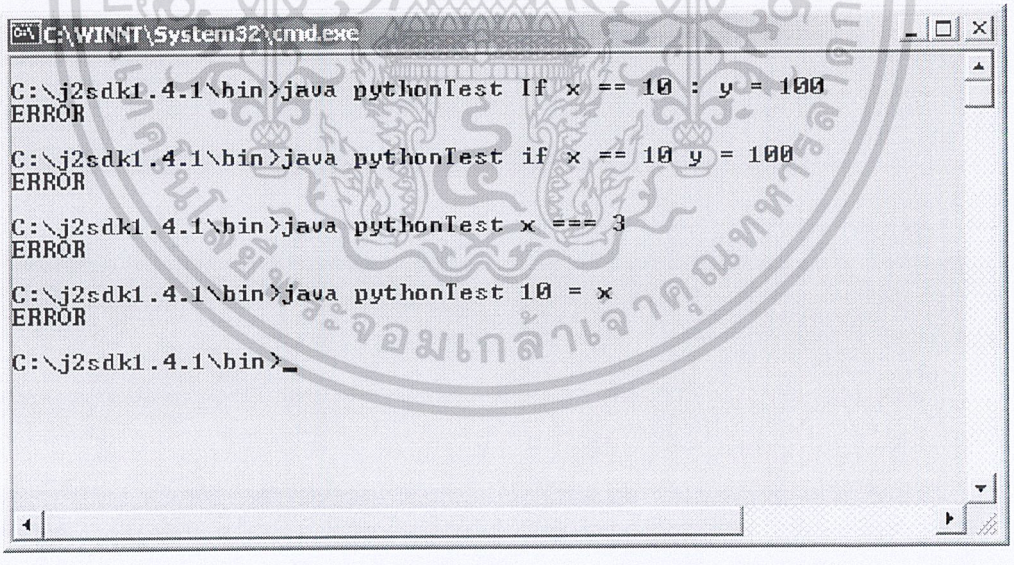
ซึ่งจากการทดลองกับประโยคต่างๆ จึงได้ผลออกมาดังต่อไปนี้



รูปที่ 5-6 แสดงตัวอย่างการทดสอบประโยคที่เป็นคำสั่งแบบตามลำดับทั่วๆ ไป



รูปที่ 5-7 แสดงตัวอย่างการทดสอบประโยคที่เป็นคำสั่งแบบเงื่อนไขและแบบวนรอบ



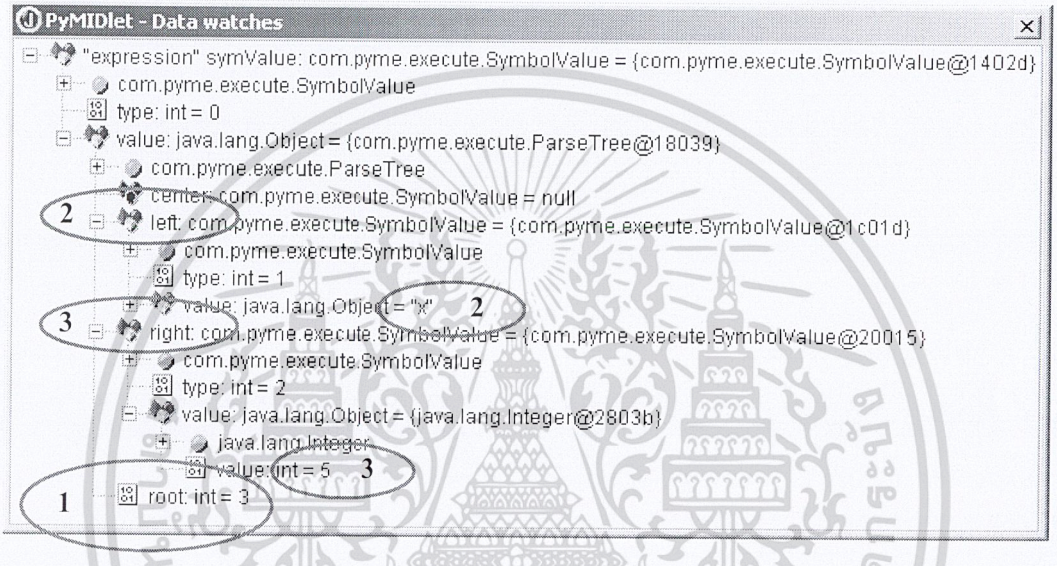
รูปที่ 5-8 แสดงตัวอย่างการทดสอบประโยคที่ไม่ถูกต้องตามบีเอ็นเอฟ

5.3 การทดสอบความถูกต้องของพาร์ซทรี

การทดสอบความถูกต้องของพาร์ซทรีนั้นเราจะกระทำโดยการเพิ่มการสร้างพาร์ซทรีเข้าไปในส่วนของการตรวจสอบประโยค ซึ่งเราจะทำการดีบัค (debug) ดูค่าตัวแปรที่ใช้เป็นพาร์ซที โดยหากประโยคมีความถูกต้องโปรแกรมจะทำการสร้างพาร์ซทรีแล้วส่งออกมารอการกระทำตามคำสั่งอีกที

ซึ่งจากการทดลองกับประโยคต่างๆ จึงได้ผลออกมาดังต่อไปนี้

5.3.1 ทดลองป้อนคำสั่งเป็น x + 5 ได้ผลเป็นดังนี้



รูปที่ 5-9 แสดงตัวอย่างการทดสอบพาร์ซทรีของประโยค x + 5

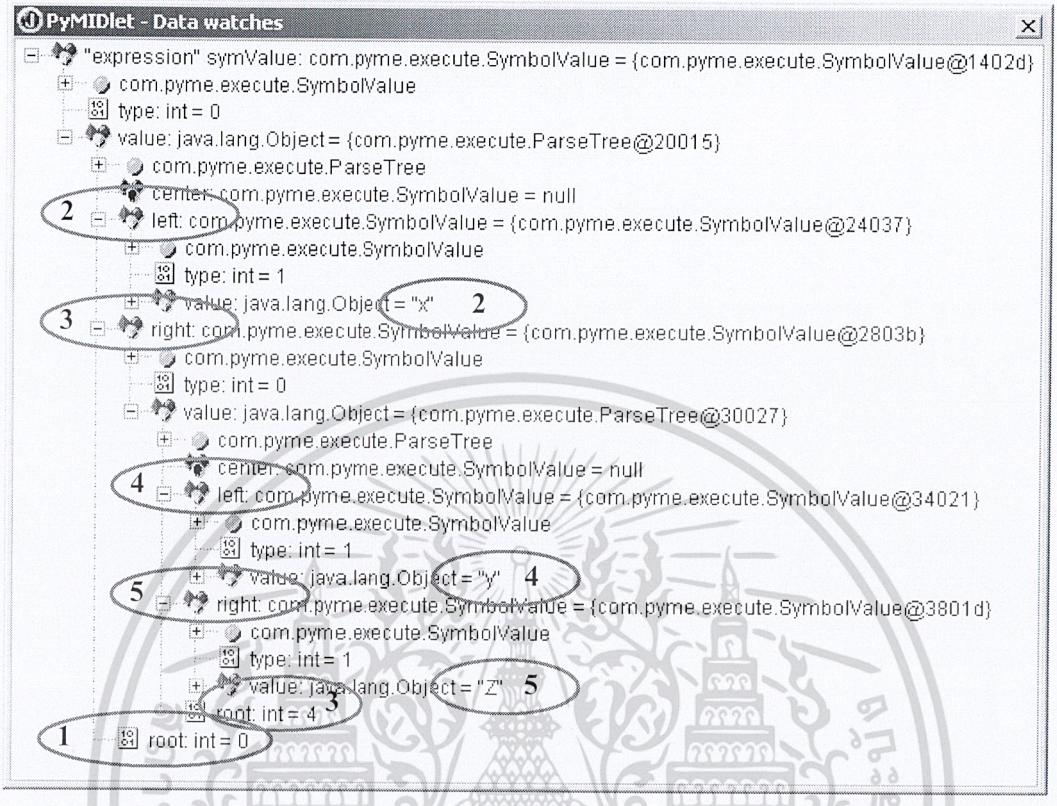
จากรูปที่ 5-9 นี้มีความหมายดังต่อไปนี้

หมายเลข 1 แสดงถึงรากของพาร์ซทรีซึ่งมีเท่ากับ 3 ซึ่งได้กำหนดไว้ว่าเป็นพาร์ซทรีของการบวกเลข

หมายเลข 2 แสดงถึงลูกทางด้านซ้ายของพาร์ซทรีหมายเลข 1 ซึ่งมีค่าเป็น “x”

หมายเลข 3 แสดงถึงลูกทางด้านขวาของพาร์ซทรีหมายเลข 1 ซึ่งมีค่าเป็น 5

5.3.2 ทดลองป้อนคำสั่งเป็น x = y - Z ได้ผลเป็นดังนี้



รูปที่ 5-10 แสดงตัวอย่างการทดสอบพาร์ซทรีของประโยค x = y - Z

จากรูปที่ 5-10 นี้มีความหมายดังต่อไปนี้

หมายเลข 1 แสดงถึงรากของพาร์ซทรีซึ่งมีเท่ากับ 0 ซึ่งได้กำหนดไว้ว่าเป็นพาร์ซทรีของคำสั่งการกำหนดค่า (assignment statement)

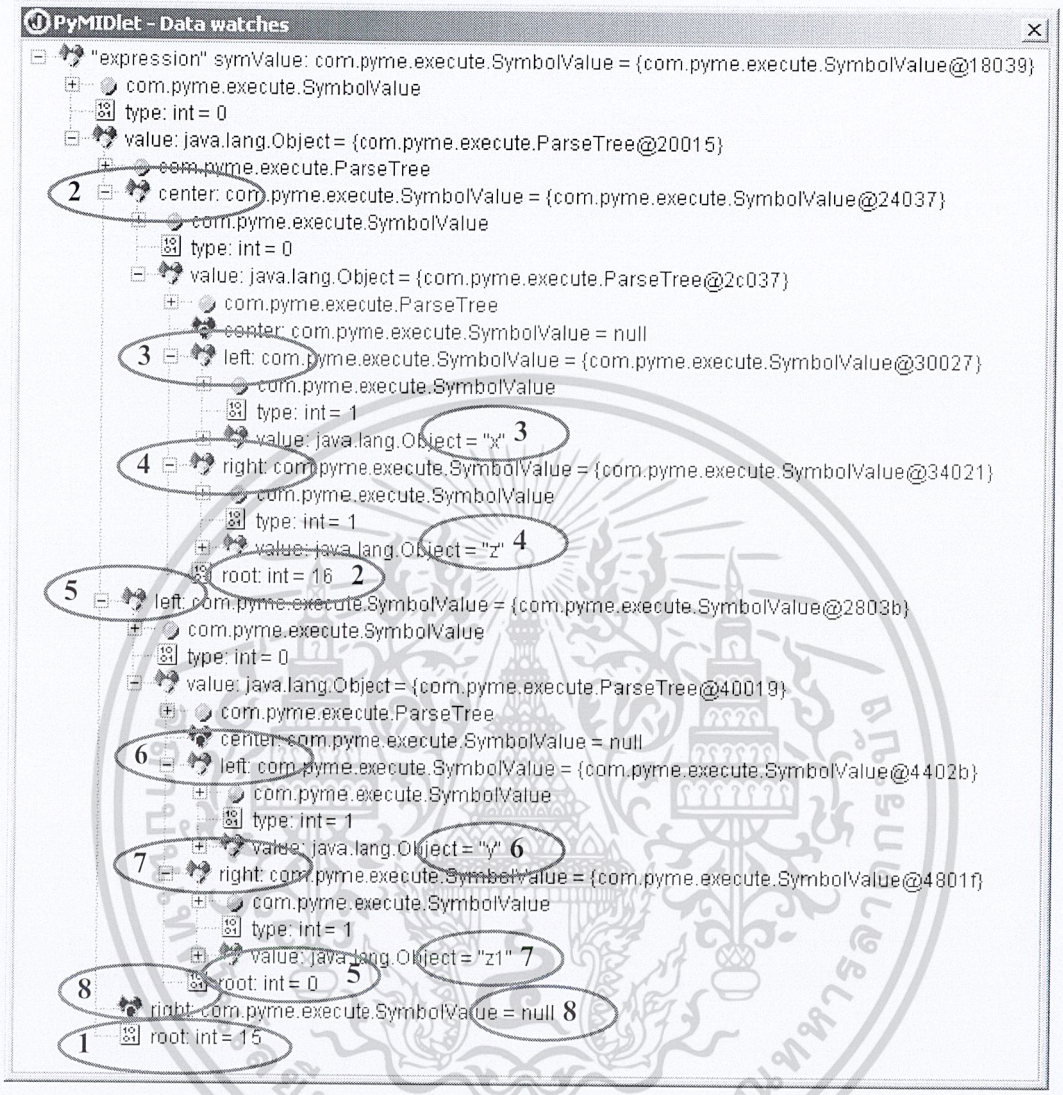
หมายเลข 2 แสดงถึงลูกทางด้านซ้ายของพาร์ซทรีหมายเลข 1 ซึ่งมีค่าเป็น “x”

หมายเลข 3 แสดงถึงลูกทางด้านขวาของพาร์ซทรีหมายเลข 1 ซึ่งมีลักษณะเป็นพาร์ซทรีซึ่งมีรากของพาร์ซทรีมีค่าเป็น 4 ซึ่งได้กำหนดไว้ว่าเป็นพาร์ซทรีของการลบเลข

หมายเลข 4 แสดงถึงลูกทางด้านซ้ายของพาร์ซทรีหมายเลข 3 ซึ่งมีค่าเป็น “y”

หมายเลข 5 แสดงถึงลูกทางด้านขวาของพาร์ซทรีหมายเลข 3 ซึ่งมีค่าเป็น “Z”

5.3.3 ทดลองป้อนคำสั่งเป็น if x == z : y = z1 ได้ผลเป็นดังนี้



รูปที่ 5-11 แสดงตัวอย่างการทดสอบพาร์ซทรีของประโยค if x == z : y = z1

จากรูปที่ 5-11 นี้มีความหมายดังต่อไปนี้

หมายเลข 1 แสดงถึงรากของพาร์ซทรีซึ่งมีเท่ากับ 15 ซึ่งได้กำหนดไว้ว่าเป็นพาร์ซทรีของคำสั่ง if statement

หมายเลข 2 แสดงถึงลูกตรงกลางของพาร์ซทรีหมายเลข 1 ซึ่งมีลักษณะเป็นพาร์ซทรี โดยจะเป็นเงื่อนไขของคำสั่ง if มีรากของพาร์ซทรีมีค่าเป็น 16 ซึ่งได้กำหนดไว้ว่าเป็นพาร์ซทรีของเงื่อนไขเท่ากับ

หมายเลข 3 แสดงถึงลูกทางด้านซ้ายของพาร์ซทรีหมายเลข 2 ซึ่งมีค่าเป็น “x”

หมายเลข 4 แสดงถึงลูกทางด้านขวาของพาร์ซทรีหมายเลข 2 ซึ่งมีค่าเป็น “z”

หมายเลข 5 แสดงถึงลูกทางด้านซ้ายของพาร์ซทรีหมายเลข 1 ซึ่งมีลักษณะเป็นพาร์ซทรีซึ่งมีรากของพาร์ซทรีมีค่าเป็น 0 ซึ่งได้กำหนดไว้ว่าเป็นพาร์ซทรีของของคำสั่งการกำหนดค่า (assignment statement)

หมายเลข 6 แสดงถึงลูกทางด้านซ้ายของพาร์ซทรีหมายเลข 5 ซึ่งมีค่าเป็น “y”

หมายเลข 7 แสดงถึงลูกทางด้านขวาของพาร์ซทรีหมายเลข 5 ซึ่งมีค่าเป็น “z1”

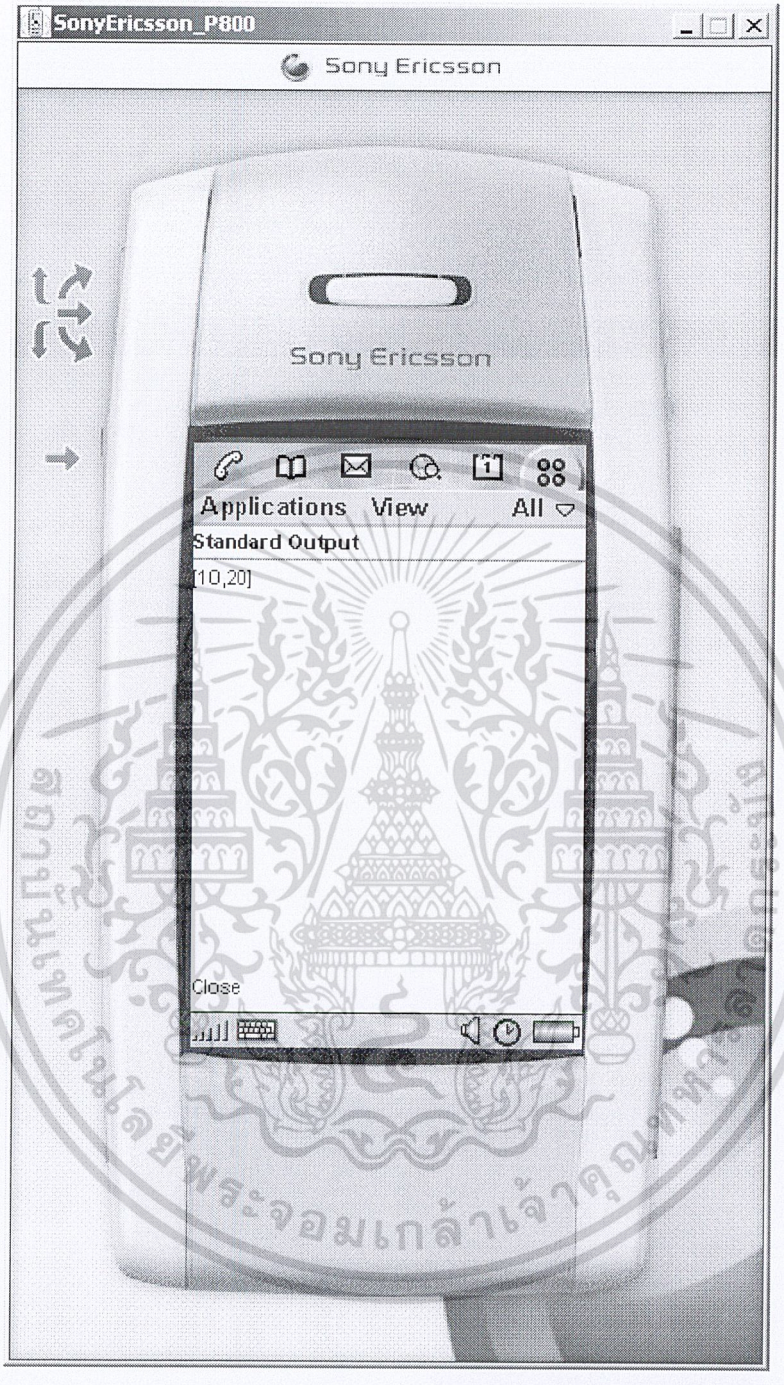
หมายเลข 8 แสดงถึงลูกทางด้านขวาของพาร์ซทรีหมายเลข 1 ซึ่งมีค่าเป็น null ซึ่งนั่นหมายความว่าคำสั่ง if นี้ไม่มี else นั่นเอง

5.4 การทดสอบความถูกต้องของการกระทำตามคำสั่ง และนำไปใช้งานบนมือถือ

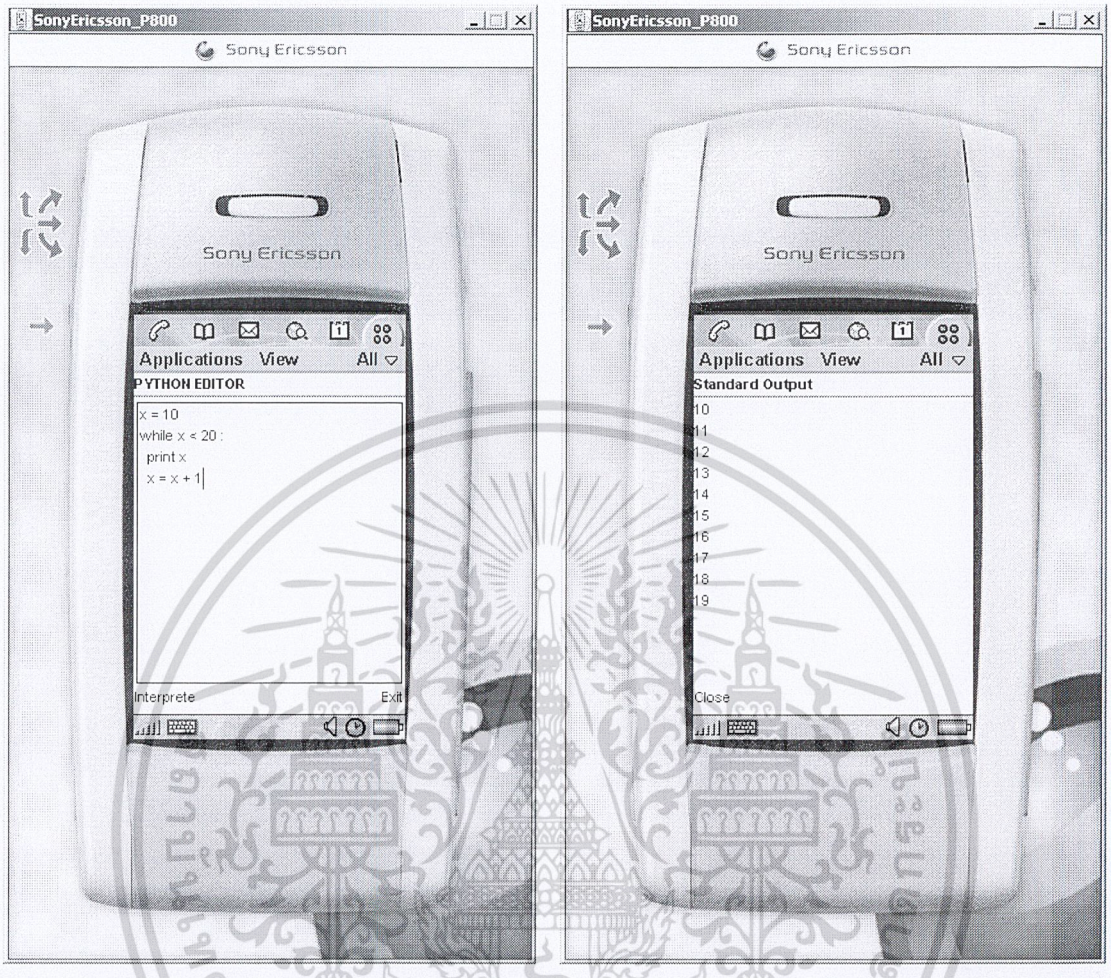
การทดสอบความถูกต้องของการกระทำตามคำสั่งพาร์ซทรีนั้นเราจะนำเอาพาร์ซทรีที่ถูกต้องทั้งหมดแล้ว มาทำการประมวลผลและแสดงผลลัพธ์ออกมา โดยเพื่อให้การทดสอบตรงกับความเป็นจริงและพร้อมที่จะนำไปใช้ได้เลย เราจะทำการทดสอบกับโปรแกรมจำลองมือถือ (mobile emulator) และนำไปทดสอบบนมือถือจริงด้วย

ซึ่งจากการทดลองกับคำสั่งต่างๆ จึงได้ผลออกมาดังต่อไปนี้

รูปที่ 5-12 แสดงตัวอย่างการป้อนสคริปต์การกำหนดค่าแล้วแสดงออกทางหน้าจอ

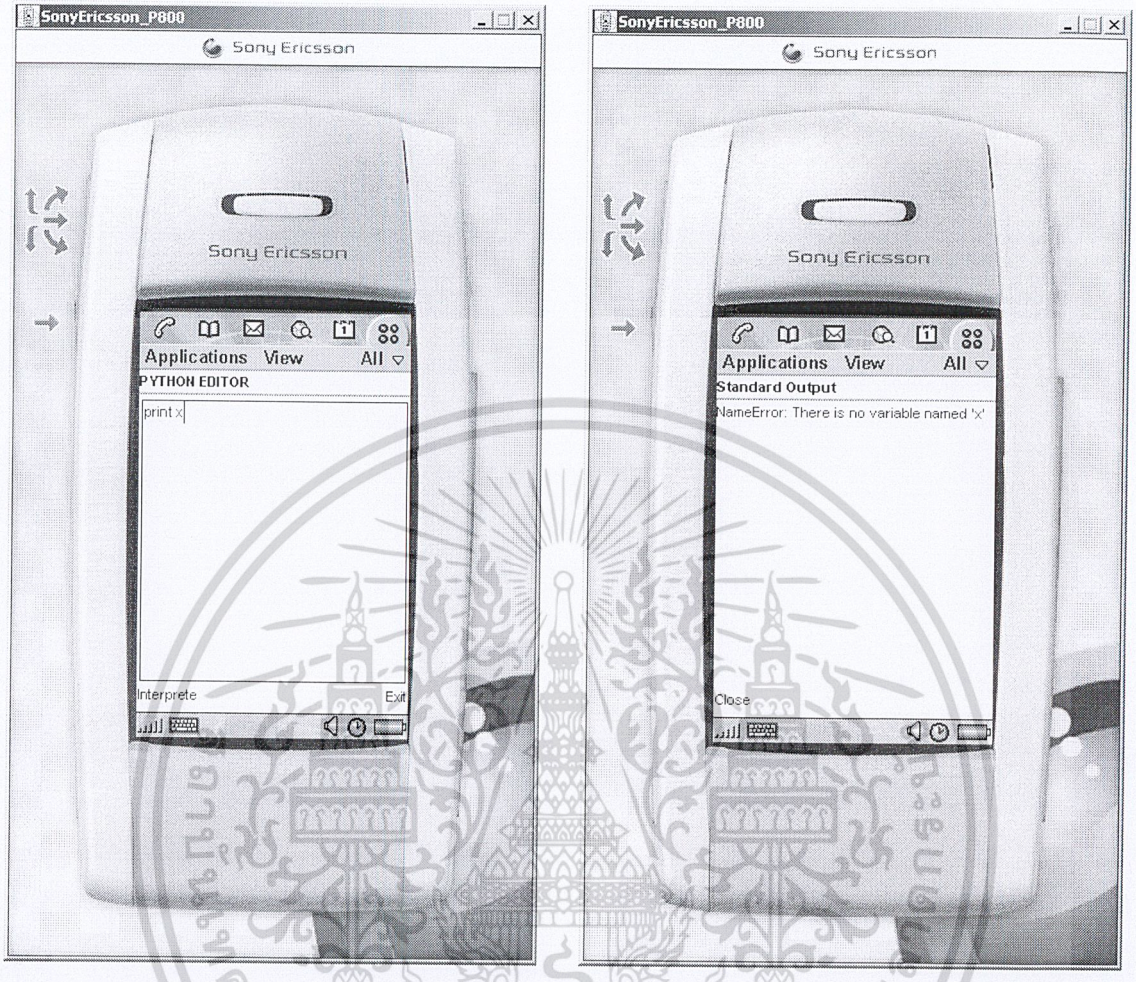


รูปที่ 5-13 แสดงผลลัพธ์จากการอินเตอร์พรีตสคริปต์การกำหนดค่าแล้วแสดงออกทางหน้าจอ



รูปที่ 5-14 แสดงตัวอย่างการป้อนและผลลัพธ์จากสคริปต์วนลูปและแสดงค่า

จากรูปที่ 5-14 เป็นการกำหนดค่าให้กับ x เป็น 10 แล้วทำการวบรอบการทำงานโดยมีเงื่อนไขว่า ถ้า x น้อยกว่า 20 ให้ทำการแสดงค่า x และเพิ่มค่า x ขึ้นอีก 1 ซึ่งก็จะได้ผลลัพธ์เป็นค่า 10 ถึง 19 ตามลำดับ แสดงออกมาทางหน้าจอ



รูปที่ 5-15 แสดงตัวอย่างการป้อนสคริปต์ที่มีความผิดพลาด

จากรูปที่ 5-15 เป็นการสั่งให้โปรแกรมแสดงค่าของ x ออกมาทางหน้าจอ แต่เนื่องจาก x ยังไม่เคยมีการกำหนดค่ามาก่อน ดังนั้นโปรแกรมจึงแสดงข้อความที่บ่งบอกถึงความผิดพลาดว่า ไม่มีตัวแปรที่ชื่อว่า 'x'

บทที่ 6
บทวิจารณ์และสรุป

6.1 บทสรุป

ปริญญานิพนธ์ฉบับนี้แสดงให้เห็นถึงการพัฒนาอินเตอร์พรีตเตอร์ไพธอนขนาดเล็ก โดยเริ่มตั้งแต่หลักการพื้นฐานต่างๆ ที่เกี่ยวข้อง ตลอดจนแนวคิดหลักในการพัฒนาโปรแกรม และแสดงให้เห็นถึงขั้นตอนในการพัฒนาโปรแกรม รวมถึงผลลัพธ์ที่ได้ ซึ่งเป็นอินเตอร์พรีตเตอร์ไพธอนขนาดเล็ก ซึ่งมีความสามารถดังต่อไปนี้

1. สามารถตรวจจับกลุ่มคำส่วนใหญ่ของภาษาไพธอนได้
2. สามารถตรวจสอบประโยคของคำสั่งส่วนใหญ่ของภาษาไพธอนได้
3. สามารถอินเตอร์พรีตคำสั่งแบบลำดับทั่วๆ ไปของภาษาไพธอนได้
4. สามารถอินเตอร์พรีตคำสั่งแบบมีเงื่อนไขของภาษาไพธอนได้
5. สามารถอินเตอร์พรีตคำสั่งแบบมีการวนรอบของภาษาไพธอนได้
6. สามารถเขียนสคริปต์และทำการอินเตอร์พรีตสคริปต์โดยทำงานอยู่บนโทรศัพท์มือถือได้
7. สามารถส่งการกระทำตามเหตุการณ์ (action event) เพื่อให้บราวเซอร์ทำการเปิดเว็ปและส่งเมลล์ได้

6.2 ข้อดีของอินเตอร์พรีตเตอร์ไพธอนขนาดเล็ก

จากความสามารถของอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กที่ได้กล่าวมาข้างต้น สามารถสรุปเป็นข้อดีของโปรแกรมได้ ดังนี้

1. สามารถทำงานได้กับมือถือใดๆ ก็ได้ที่รองรับการทำงานของ java 2 micro edition (J2ME) เนื่องจากว่าได้ใช้เพียงคำสั่งมาตราฐานของ j2me ในการพัฒนาเท่านั้น
2. สามารถนำอินเตอร์พรีตเตอร์นี้ร่วมกับโปรแกรมอื่นๆ ที่อาจจะไม่ใช่บราวเซอร์ก็ได้ เนื่องจากได้พัฒนาไว้ในรูปแบบของ class ซึ่งสามารถนำไปใช้ได้ทันที
3. สามารถขยายเพิ่มเติมความสามารถในตัวภาษาได้และในตัวอินเตอร์พรีตเตอร์ได้

6.3 ข้อเสียของอินเตอร์พรีตเตอร์ไพธอนขนาดเล็ก

เนื่องจากข้อจำกัดของตัวภาษาที่นำมาพัฒนาและสร้างอินเตอร์พรีตเตอร์นี้ยังมีข้อจำกัดอยู่มาก อีกทั้งอุปกรณ์มือถือเองยังมีข้อจำกัดอยู่มากมายเช่น ขนาดของหน้าจอการแสดงผลที่ไม่ใหญ่มากนัก หน่วยความจำที่มีค่อนข้างน้อย และคุณสมบัติในการทำงานของ j2me ยังไม่สามารถที่จะจัดการกับตัวมือถือได้อย่างเต็มที่ ทำให้อินเตอร์พรีตเตอร์ไพธอนขนาดเล็กที่นำไปใช้กับบราวเซอร์บนมือถือนั้นยังมีปัญหาและข้อเสีย ดังนี้

1. ไม่มีชนิดข้อมูลแบบ float
2. ไม่สามารถสร้างฟังก์ชั่นการทำงานที่จะทำการติดต่อกับ การส่งข้อความสั้น (short message service), การโทรออก, อินฟาเรด และ บลูธูท (Bluetooth) ได้
3. ยังมีความช้าในการอินเตอร์พรีตอยู่

6.4 แนวทางการพัฒนา

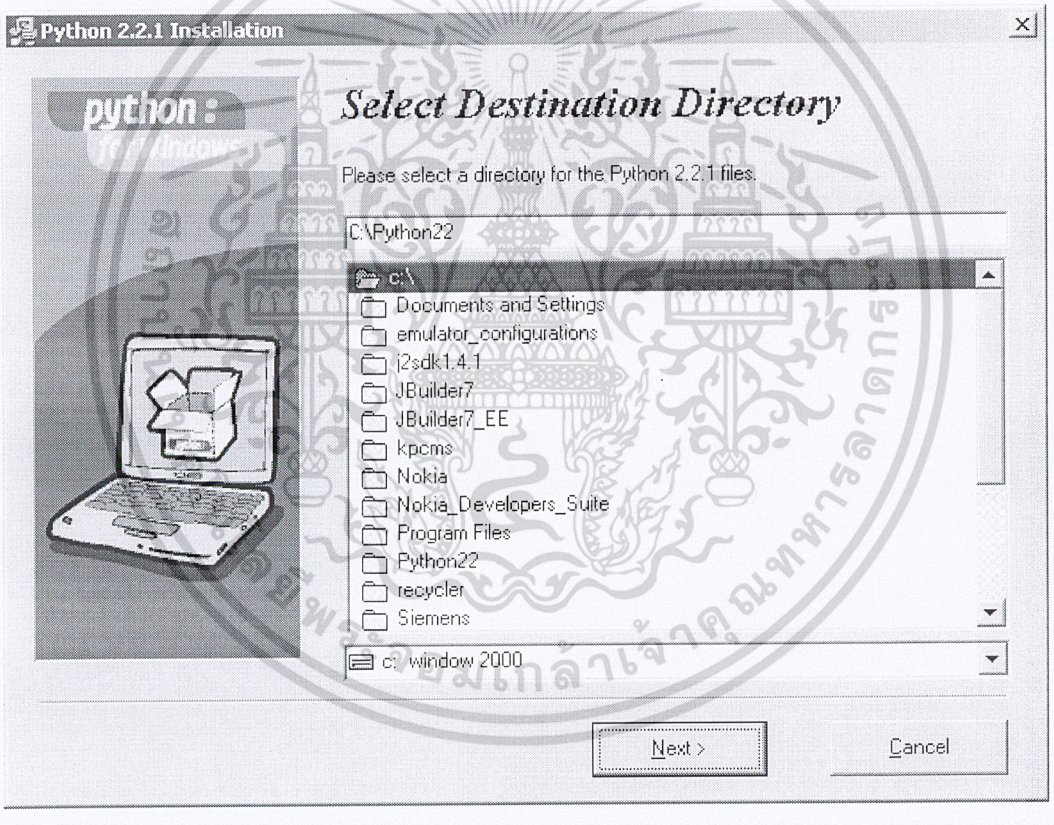
จากปัญหาและข้อเสียที่ได้กล่าวถึงแล้วข้างต้น จะเห็นได้ว่าอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กนั้น ยังมีสิ่งที่ต้องทำการพัฒนาต่อไปอีกมากมาย เพื่อให้ได้อินเตอร์พรีตเตอร์ที่สมบูรณ์ และตรงกับความต้องการในการใช้งานของผู้ใช้มากที่สุด โดยสิ่งที่ควรได้รับการพัฒนาต่อ มีดังนี้

1. เลือกภาษาที่สามารถเขียนโปรแกรมบนมือถือได้ และมีประสิทธิภาพมากกว่า java 2 micro edition ซึ่งอาจจะเป็นภาษา Personal Java หรือภาษา C++ บนโทรศัพท์มือถือ ซึ่งจะทำมีความสามารถในการเข้าถึงฮาร์ทแวร์ของตัวมือถือได้มากขึ้น
2. เพิ่มชนิดของข้อมูล และคำสั่งต่างๆ ที่จำเป็นสำหรับบราวเซอร์บนมือถือ
3. เพิ่มคำสั่งที่ทำงานเกี่ยวกับเวลา เพื่อให้สคริปต์มีประสิทธิภาพมากขึ้น

ภาคผนวก ก.
การติดตั้งและใช้งานภาษาไพธอน

ขั้นตอนการติดตั้งโปรแกรม python 2.2.1

1. ดาวน์โหลดชุดติดตั้งโปรแกรมไพธอนได้จาก www.python.org โดยไม่ต้องเสียค่าใช้จ่ายใดๆ ซึ่งจะได้ไฟล์ Python-2.2.1.exe ซึ่งเป็นไฟล์ที่ใช้ติดตั้งภาษาไพธอนเวอร์ชั่น 2.2.1 บนระบบปฏิบัติการวินโดว์
2. ทำการรันไฟล์ Python-2.2.1.exe เพื่อทำการติดตั้ง โดยจะแสดงหน้าจอดังนี้



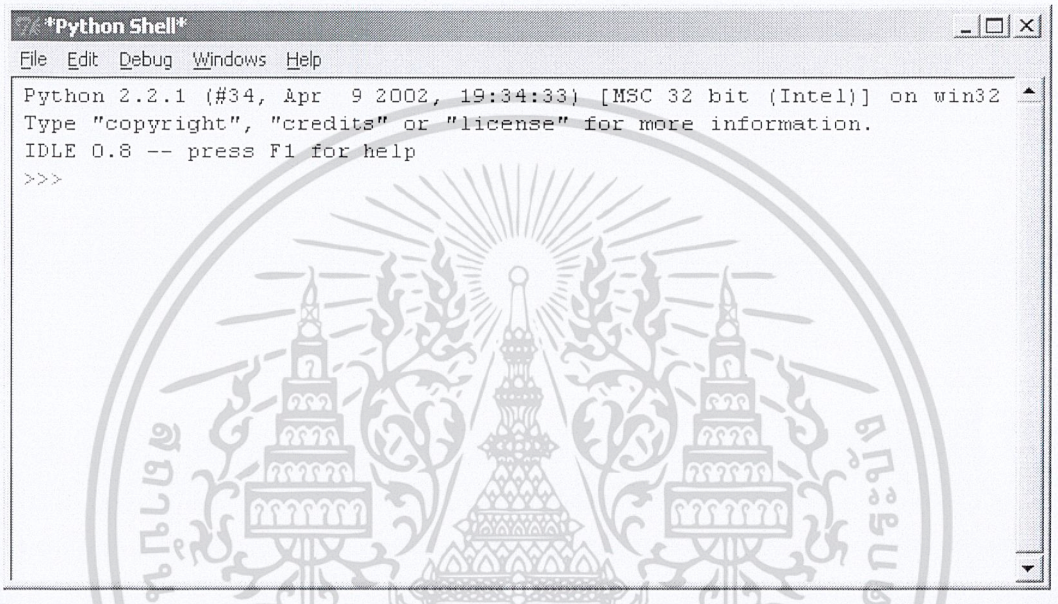
รูปที่ ก-1 แสดงหน้าต่างการติดตั้งโปรแกรมไพธอนเวอร์ชั่น 2.2.1

3. ทำการเลือกไดเรคทอรีที่ต้องการจะติดตั้ง และปฏิบัติตามขั้นตอนเหมือนการลงโปรแกรมทั่วๆ ไปก็จะเป็นอันเสร็จการติดตั้งโปรแกรมไพธอนเวอร์ชั่น 2.2.1

การใช้งานภาษาไพธอน

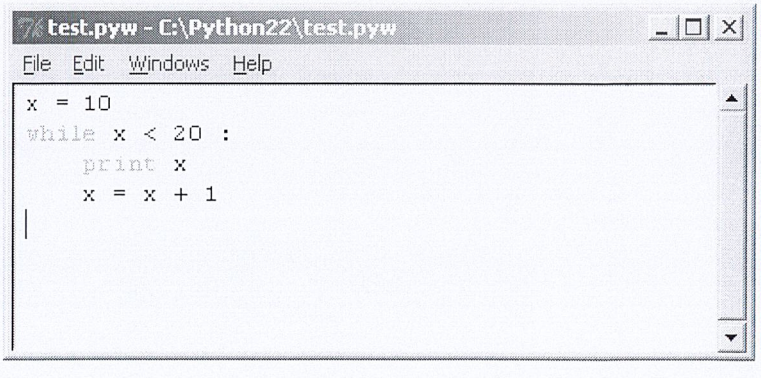
เนื่องจากโครงงานวิจัยนี้เป็นการพัฒนาอินเตอร์พรีตเตอร์ที่อ้างอิงภาษาไพธอนเป็นส่วนใหญ่ ควรจะเข้าใจการใช้งานของภาษาไพธอนบ้างเล็กน้อย โดยจะใช้เพื่อทดลองคำสั่งต่างๆ ของภาษาไพธอน ซึ่งมีวิธีการใช้งานดังนี้

1. คลิกที่ Start → Programs → Python 2.1 → IDLE (Python GUI) ซึ่งจะปรากฏหน้าต่างดังรูป ก-2



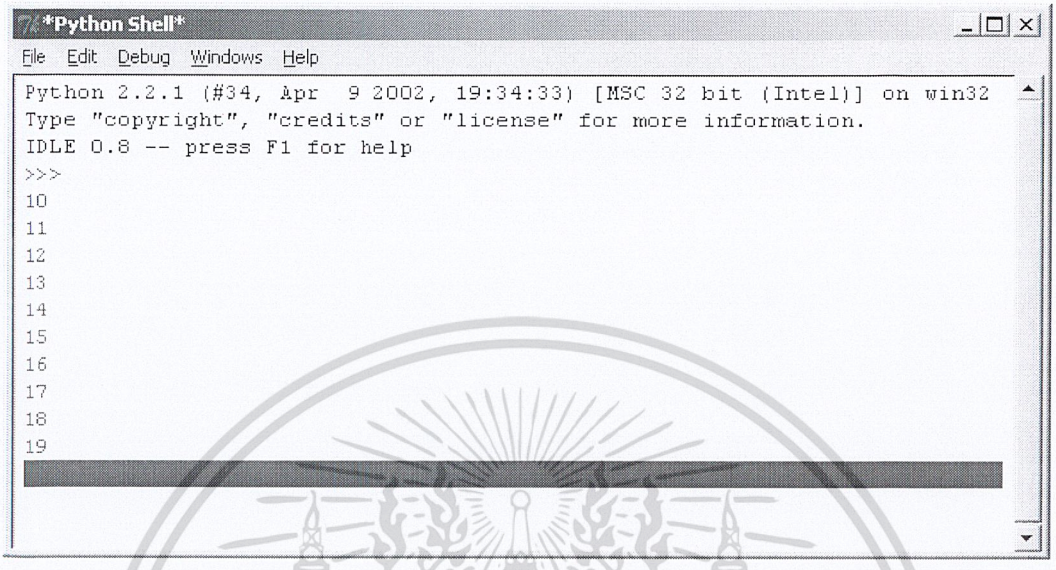
รูปที่ ก-2 แสดงหน้าต่างโปรแกรมไพธอน 2.2.1

2. สามารถที่จะทำการพิมพ์คำสั่งใดๆ ก็ได้ในภาษาไพธอน โดยโปรแกรมจะทำการอินเตอร์พรีตทันที
3. สามารถที่จะนำไฟล์สคริปต์ที่เขียนไว้แล้วมาทำการทดสอบก็ได้ โดยเลือกที่เมนู File → Open... แล้วทำการเลือกสคริปต์ที่ต้องการทำการรัน ซึ่งจะปรากฏหน้าจอดังรูปที่ ก-3



รูปที่ ก-3 แสดงหน้าต่างการเขียนสคริปต์เพื่อทำการรัน

4. หากต้องการทำการรันสคริปต์ที่เขียนไว้แล้วสามารถทำได้โดยการเลือกที่เมนู Edit → Run script ซึ่งหากทดลองรันสคริปต์จากรูป ก-3 โปรแกรมก็จะทำการอินเตอร์พรีตแล้วแสดงผลดังรูปที่ ก-4



รูปที่ ก-4 แสดงผลลัพธ์จากการรันสคริปต์จากรูป ก-3

ภาคผนวก ข.

ไวยากรณ์ของภาษาไพธอน (Python Grammar)

ไวยากรณ์ของภาษาไพธอนทั้งหมด ซึ่งจะนำไว้อ้างอิง และช่วยในการออกแบบเร็กกูล่าร์เอ็กเพสชั่นและบีเอ็นเอฟสำหรับใช้กับอินเตอร์พรีตเตอร์ไพธอนขนาดเล็กสำหรับบราวเซอร์ที่โปรแกรมได้บนโทรศัพท์มือถือ

```
identifier    ::=    (letter|"_") (letter | digit | "_")*
letter        ::=    lowercase | uppercase
lowercase     ::=    "a"..."z"
uppercase     ::=    "A"..."Z"
digit         ::=    "0"..."9"
stringliteral ::=    [stringprefix](shortstring | longstring)
stringprefix  ::=    "r" | "u" | "ur" | "R" | "U" | "UR" | "Ur" | "uR"
shortstring   ::=    "'" shortstringitem* "'" | '"' shortstringitem* '"'
longstring    ::=    "'''" longstringitem* "'''" | '"""' longstringitem* '"""'
shortstringitem      ::=    shortstringchar | escapeseq
longstringitem       ::=    longstringchar | escapeseq
shortstringchar      ::=    <any ASCII character except "\" or newline or the
                            quote>
longstringchar       ::=    <any ASCII character except "\">
escapeseq     ::=    "\" <any ASCII character>
longinteger   ::=    integer ("l" | "L")
integer       ::=    decimalinteger | octinteger | hexinteger
decimalinteger       ::=    nonzerodigit digit* | "0"
octinteger    ::=    "0" octdigit+
hexinteger    ::=    "0" ("x" | "X") hexdigit+
nonzerodigit         ::=    "1"..."9"
octdigit      ::=    "0"..."7"
hexdigit      ::=    digit | "a"..."f" | "A"..."F"
floatnumber          ::=    pointfloat | exponentfloat
pointfloat    ::=    [intpart] fraction | intpart "."
```

```
exponentfloat       ::=  (intpart | pointfloat)
intpart             ::=  digit+
fraction            ::=  "." digit+
exponent            ::=  ("e" | "E") ["+" | "-"] digit+
imagnumber          ::=  (floatnumber | intpart) ("j" | "J")
atom                ::=  identifier | literal | enclosure
enclosure           ::=  parenth_form | list_display | dict_display | string_conversion
literal             ::=  stringliteral | integer | longinteger | floatnumber | imagnumber
parenth_form        ::=  "(" [expression_list] ")"
list_display        ::=  "[" [listmaker] "]"
listmaker           ::=  expression ( list_for | ( "," expression)* [","] )
list_iter           ::=  list_for | list_if
list_for            ::=  "for" expression_list "in" testlist [list_iter]
list_if             ::=  "if" test [list_iter]
dict_display        ::=  "\{" [key_datum_list] "\}"
key_datum_list      ::=  key_datum ("," key_datum)* [","]
key_datum           ::=  expression ":" expression
string_conversion   ::=  "`" expression_list "`"
primary             ::=  atom | attributeref | subscription | slicing | call
attributeref        ::=  primary "." identifier
subscription        ::=  primary "[" expression_list "]"
slicing             ::=  simple_slicing | extended_slicing
simple_slicing      ::=  primary "[" short_slice "]"
extended_slicing    ::=  primary "[" slice_list "]"
slice_list          ::=  slice_item ("," slice_item)* [","]
slice_item          ::=  expression | proper_slice | ellipsis
proper_slice        ::=  short_slice | long_slice
short_slice         ::=  [lower_bound] ":" [upper_bound]
long_slice          ::=  short_slice ":" [stride]
lower_bound         ::=  expression
upper_bound         ::=  expression
stride              ::=  expression
ellipsis            ::=  "..."
call                ::=  primary "(" [argument_list [","]] ")"
```

```
argument_list          ::=  positional_arguments ["," keyword_arguments]
                            ["," "*" expression] ["," "**" expression]
                            | keyword_arguments ["," "*" expression]
                            ["," "**" expression]
                            | "*" expression ["," "**" expression]
                            | "**" expression

positional_arguments   ::=  expression ("," expression)*
keyword_arguments      ::=  keyword_item ("," keyword_item)*
keyword_item           ::=  identifier "=" expression
power          ::=  primary ["**" u_expr]
u_expr         ::=  power | "-" u_expr | "+" u_expr | "\~" u_expr
m_expr         ::=  u_expr | m_expr "*" u_expr | m_expr "//" u_expr
                    | m_expr "/" u_expr | m_expr "\%" u_expr
a_expr         ::=  m_expr | a_expr "+" m_expr | a_expr "-" m_expr
shift_expr     ::=  a_expr | shift_expr ( "<<" | ">>" ) a_expr
and_expr       ::=  shift_expr | and_expr "\;SPMamp;" shift_expr
xor_expr       ::=  and_expr | xor_expr "\textasciicircum" and_expr
or_expr        ::=  xor_expr | or_expr "|" xor_expr
comparison     ::=  or_expr ( comp_operator or_expr )*
comp_operator          ::=  "<" | ">" | "==" | ">=" | "<=" | "<>" | "!=" | "is" ["not"]
                            | ["not"] "in"
expression     ::=  or_test | lambda_form
or_test        ::=  and_test | or_test "or" and_test
and_test       ::=  not_test | and_test "and" not_test
not_test       ::=  comparison | "not" not_test
lambda_form            ::=  "lambda" [parameter_list]: expression
expression_list        ::=  expression ( "," expression )* [","]
simple_stmt            ::=  expression_stmt | assert_stmt | assignment_stmt |
                            augmented_assignment_stmt | pass_stmt | del_stmt |
                            print_stmt | return_stmt | yield_stmt | raise_stmt |
                            break_stmt | continue_stmt | import_stmt |
                            global_stmt | exec_stmt
```

```
expression_stmt     ::=     expression_list
assert_statement    ::=     "assert" expression ["," expression]
assignment_stmt     ::=     (target_list "=")+ expression_list
target_list  ::=    target ("," target)* [","]
target       ::=    identifier | "(" target_list ")" | "[" target_list "]" | attributeref |
                    subscription | slicing
augmented_assignment_stmt       ::=     target augop expression_list
augop        ::=    "+=" | "-=" | "*=" | "/=" | "\%=" | "**=" | ">>=" | "<<=" | "\&=" |
                    "\textasciicircum=" | "|="
pass_stmt    ::=    "pass"
del_stmt     ::=    "del" target_list
print_stmt   ::=    "print" ( \optionalexpression ("," expression)* \optional"," |
                    ">\code>" expression \optional("," expression)+ \optional"," )
return_stmt  ::=    "return" [expression_list]
yield_stmt   ::=    "yield" expression_list
raise_stmt   ::=    "raise" [expression ["," expression ["," expression]]]
break_stmt   ::=    "break"
continue_stmt       ::=     "continue"
import_stmt         ::=     "import" module ["as" name]
                            ( "," module ["as" name] )* |
                            "from" module "import" identifier ["as" name]
                            ( "," identifier ["as" name] )* |
                            "from" module "import" "*"
module       ::=    (identifier ".")* identifier
global_stmt  ::=    "global" identifier ("," identifier)*
exec_stmt    ::=    "exec" expression ["in" expression ["," expression]]
compound_stmt       ::=     if_stmt | while_stmt | for_stmt | try_stmt | funcdef |
                            classdef
suite        ::=    stmt_list NEWLINE | NEWLINE INDENT statement+ DEDENT
statement    ::=    stmt_list NEWLINE | compound_stmt
stmt_list    ::=    simple_stmt (";" simple_stmt)* [";"]
if_stmt      ::=    "if" expression ":" suite ( "elif" expression ":" suite )*
                    ["else" ":" suite]
```

```
while_stmt ::=          "while" expression ":" suite ["else" ":" suite]
for_stmt   ::=          "for" target_list "in" expression_list ":" suite ["else" ":" suite]
try_stmt   ::=          try_exc_stmt | try_fin_stmt
try_exc_stmt            ::=         "try" ":" suite ("except" [expression ["," target]] ":"
                                    suite)+ ["else" ":" suite]
try_fin_stmt            ::=         "try" ":" suite "finally" ":" suite
funcdef    ::=          "def" funcname "(" [parameter_list] ")" ":" suite
parameter_list          ::=         (defparameter ",")* ("*" identifier [, "**" identifier] |
                                    "**" identifier | defparameter [","])
defparameter            ::=         parameter ["=" expression]
sublist    ::=          parameter ("," parameter)* [","]
parameter  ::=          identifier | "(" sublist ")"
funcname   ::=          identifier
classdef   ::=          "class" classname [inheritance] ":" suite
inheritance ::=         "(" [expression_list] ")"
classname  ::=          identifier
file_input  ::=         (NEWLINE | statement)*
interactive_input       ::=         [stmt_list] NEWLINE | compound_stmt NEWLINE
eval_input  ::=         expression_list NEWLINE*
input_input ::=         expression_list NEWLINE
```

บรรณานุกรม

[1] วีระศักดิ์ ซึงถาวร : "Fundamental of Java Programming Volume I", SUM System Company Limited, 2543.

[2] Mark Lutz and David Ascher : "Learning Python", O'reilly & Associates Inc., CA, 1999.

[3] H.M. Deitel, P.J. Deitel, J.P. Liperi and B.A. Wiedermann : "Python How To Program", Prentice-Hall Inc., Upper saddle River, NJ 07458, 2002.

[4] Vartan Piroumian : "Wireless J2ME Platform Programming", Prentice Hall PTR, One Lak Street, Upper Saddle River, NJ 07458, 2002.

[5] John W. Muchow : "Core J2ME Technology & MIDP", Prentice Hall PTR, One Lak Street, Upper Saddle River, NJ 07458, 2002.

[6] "Python Language Website", http://www.python.org

[7] Guido van Rossum and Fred L. Drake, Jr : "Python Reference Manual", http://www.python.org/doc/current/ref/

[8] "Bumble-Bee Software – Parser Generator – YACC and Lex for Windows" http://www.bumblebeesoftware.com