

# ภาษาโปรแกรมมิ่งไพธอน Python programming language

เอกวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยนเรศวร



## รายชื่อผู้จัดทำ

ชื่อ	รหัสนิสิต	หน้าที่ดำเนินการ
นายฉัตรชัย ดำดี	46320388	แบบฝึกหัด
นายทรงยศ คชนิล	46320511	จัดทำเอกสารการเรียน
นางสาวธัญญากร แก้วประสงค์	46320610	ผู้ช่วยสอน
นายประจักษ์ เจตะภัย	46320693	ผู้ช่วยสอนและแบบฝึกหัด
นายมารุต จันทร์บัว	46320818	แบบฝึกหัด
นางสาวศรินยา อยู่สุขดี	46320925	ผู้ช่วยสอน
นางสาวณัฐนิชา คงประกอบ	46321097	ผู้ช่วยสอน
นายอุทิศ ศักดิ์สิทธิ์	46321139	ผู้ช่วยสอนและแบบฝึกหัด
นายอรรณพ สุวัฒนพิเศษ	46321150	ผู้สอนและจัดทำเอกสารเรียน



# คำนำ

ภาษาไพธอนเป็นภาษาที่ได้รับความนิยมอย่างมากในปัจจุบันเนื่องจากความสามารถที่สูง, การเรียนรู้ที่รวดเร็ว, การเขียนระบบที่เข้าใจง่าย และสามารถขยายขีดความสามารถในการสร้างโปรแกรมและซอฟต์แวร์ที่สูงมากขึ้นตลอดเวลา ทางทีมผู้จัดทำจึงเล็งเห็นว่าควรนำความรู้ ความเข้าใจในการเขียนโปรแกรมด้วยภาษาไพธอนมาเผยแพร่ ด้วยจะได้ผู้อื่นได้รับความรู้และได้เข้าถึงภาษาที่เขียนใจง่าย, ทำงานรวดเร็ว และสามารถสร้างสรรค์งานได้อย่างมีความสามารถสูง อีกทั้งซอฟต์แวร์ที่ใช้สร้างโปรแกรมและซอฟต์แวร์ด้วยภาษาไพธอนนั้นมีทั้งแจกฟรี, รหัสเปิด และเชิงธุรกิจ ซึ่งมีขีดความสามารถที่แตกต่างกัน แต่ถึงแม้จะเป็นซอฟต์แวร์ที่ใช้เขียนโปรแกรมด้วยภาษาไพธอนจะแจกฟรี หรือเป็นรหัสเปิด ก็ไม่ได้ด้อยไปกว่าเชิงธุรกิจเลย จึงเป็นทางเลือกที่ดีที่จะศึกษาเป็นทางเลือกอีกทางหนึ่งนอกเหนือจากภาษาอื่น ๆ ที่ได้รับความนิยมอยู่แล้ว

ทางทีมงานจึงหวังว่าท่านผู้ที่นำเอกสารนี้ไปใช้ในการศึกษาจะได้รับประโยชน์สูงสุดในการเขียนโปรแกรมและซอฟต์แวร์ด้วยภาษาไพธอน

*ทีมผู้จัดทำ*



# สารบัญ

<b>1</b>	<b>แนะนำภาษาไพธอน</b>	<b>13</b>
1.1	ประวัติ . . . . .	13
1.1.1	Python 1.0 . . . . .	13
1.1.2	Python 2.0 . . . . .	14
1.1.3	อนาคต . . . . .	14
1.2	หลักปรัชญาของภาษาไพธอน . . . . .	15
1.3	Language Evaluation Criteria . . . . .	15
1.4	ข้อเด่นของภาษาไพธอน . . . . .	15
1.5	Category และ Application Domains . . . . .	16
1.5.1	Web และ Internet Development . . . . .	16
1.5.2	Database Access . . . . .	16
1.5.3	Desktop GUI . . . . .	17
1.5.4	Scientific และ Numeric computation . . . . .	17
1.5.5	Education . . . . .	17
1.5.6	Network programming . . . . .	17
1.5.7	Software builder และ Testing . . . . .	17
1.5.8	Game และ 3D Graphics Rendering . . . . .	17
1.6	ซอฟต์แวร์ที่เขียนด้วยไพธอน . . . . .	17
1.7	ตัวอย่างความสำเร็จของไพธอน . . . . .	18
<b>2</b>	<b>การแสดงผลเบื้องต้น (Printing)</b>	<b>21</b>
<b>3</b>	<b>การตั้งตัวชื่อแปร และคำสงวน (Reserved word หรือ Keywords)</b>	<b>23</b>
3.1	การตั้งตัวชื่อแปร . . . . .	23
3.2	คำสงวนในการใช้งาน (Reserved words, Keywords) . . . . .	23
<b>4</b>	<b>การคำนวณทางคณิตศาสตร์ (Arithmetic Mathematics)</b>	<b>25</b>
4.1	การคำนวณพื้นฐาน (normal arithmetic operators) . . . . .	25
4.2	การคำนวณผ่านฟังก์ชันภายใน (Built-in Math Functions) . . . . .	26
4.2.1	การหาค่าสัมบูรณ์ (absolute value) . . . . .	26
4.2.2	จำนวนที่น้อยที่สุด และมากที่สุดในกลุ่ม (smallest or largest values) . . . . .	26
4.2.3	กำหนดจำนวนตัวเลขทศนิยม (specified number of digits) . . . . .	27
4.2.4	หาผลรวมทั้งหมดในชุดข้อมูล (adds numbers in a sequence.) . . . . .	27
4.2.5	ช่วงของข้อมูลตัวเลข (range of numbers.) . . . . .	27

<b>5</b>	<b>ชนิดของตัวแปร (Data type)</b>	<b>29</b>
5.1	ตัวเลข (Numbers)	29
5.1.1	จำนวนเต็ม(Integers)	29
5.1.2	จำนวนจริง (Floating-point numbers)	30
5.1.3	จำนวนเชิงซ้อน (Complex Numbers)	31
5.2	ชนิดข้อมูลแบบการรวมกลุ่มข้อมูล (Collection Data Types)	32
5.2.1	ลิสต์ (List)	32
5.2.2	ดิกชันนารี (Dictionary หรือ Groupings of Data Indexed by Name)	33
5.2.3	ทับเปิ้ล (Tuples) และ อนุกรม (Sequences)	34
5.2.4	เซต (Sets)	34
5.2.5	ฟังก์ชันที่น่าสนใจเกี่ยวข้องกับลิสต์และดิกชันนารี	35
5.3	สายอักขระ (String หรือ Array of Characters)	39
5.3.1	ฟังก์ชันที่น่าสนใจเกี่ยวข้องกับสายอักขระ	42
<b>6</b>	<b>การเปรียบเทียบ (Comparisons)</b>	<b>45</b>
<b>7</b>	<b>นิพจน์ทางตรรกศาสตร์ (Boolean Expressions)</b>	<b>47</b>
7.1	AND (และ)	47
7.2	OR (หรือ)	47
7.3	NOT (ไม่)	48
<b>8</b>	<b>ช่วงของการทำงาน (Statement block) และ ช่วงชีวิตของตัวแปร (Life time หรือ Variable scope)</b>	<b>49</b>
8.1	ช่วงของการทำงาน (Statement block)	49
8.2	ช่วงชีวิตของตัวแปร (Life time หรือ Variable scope)	49
<b>9</b>	<b>การควบคุมทิศทางของโปรแกรม (Control flow, Flow of Control หรือ Alternately)</b>	<b>51</b>
9.1	การตัดสินใจ (Decisions, Choice หรือ Selection)	51
9.1.1	if Statements	51
9.1.2	switch Statements	53
9.2	การวนทำซ้ำ (Loop)	53
9.2.1	while Statements	53
9.2.2	for Statements	53
9.2.3	pass, break, continue และ else Clauses Statements	54
9.2.4	do-while Statements	55
9.3	การจัดการความผิดปกติของโปรแกรม (Error Checking)	55
9.3.1	assert Statements	55
9.3.2	try-except และ raise Statements (Exception handling)	56
<b>10</b>	<b>การสร้างฟังก์ชัน (Defined Function)</b>	<b>59</b>
10.1	การรับค่าของฟังก์ชัน, คืนค่ากลับ และค่ามาตรฐานของการรับค่า	60
10.2	ตัวแปรแบบ Global (ทั่วไป) และ Local (เฉพาะส่วน)	62
<b>11</b>	<b>การใส่ข้อมูลผ่านคีย์บอร์ด (Input Data from Keyboard)</b>	<b>63</b>



<b>ก</b>	<b>เรื่องที่ห้ามลืมใน Python</b>	<b>67</b>
<b>ข</b>	<b>การติดตั้ง Python, wxPython และ Stani's Python Editor</b>	<b>69</b>
	ข.1 การติดตั้ง Python . . . . .	69
	ข.2 การติดตั้ง wxPython . . . . .	72
	ข.3 การติดตั้ง Stani's Python Editor . . . . .	75
<b>ค</b>	<b>อธิบายส่วนต่าง ๆ พอสั่งเซปของโปรแกรม SPE</b>	<b>79</b>
	ค.0.1 Sidebar . . . . .	80
	ค.0.2 Source . . . . .	81
	ค.0.3 Tools . . . . .	81
<b>ง</b>	<b>การเขียน, Debug และสั่งให้โปรแกรมทำงาน</b>	<b>85</b>
	ง.1 การเขียนโปรแกรมใน SPE และสั่งให้โปรแกรมทำงาน . . . . .	85
	ง.2 การ Debug โปรแกรม . . . . .	86
<b>จ</b>	<b>ข้อมูลอ้างอิง</b>	<b>87</b>



# สารบัญรูป

1.1	Guido van Rossum . . . . .	14
ข.1	เลือกดาวน์โหลด Python 2.4 สำหรับ Windows . . . . .	69
ข.2	ขั้นตอนที่ 1 : ดับเบิลคลิกที่ไฟล์ติดตั้ง . . . . .	70
ข.3	ขั้นตอนที่ 2 : เลือก "Install for all users" . . . . .	70
ข.4	ขั้นตอนที่ 3 : ให้เลือกที่ติดตั้งที่ C:\Python24\ . . . . .	70
ข.5	ขั้นตอนที่ 4 : เลือกติดตั้งทุกตัวเลือก . . . . .	71
ข.6	ขั้นตอนที่ 5 : ดำเนินการติดตั้ง . . . . .	71
ข.7	ขั้นตอนที่ 6 : เสร็จสิ้นการติดตั้ง . . . . .	71
ข.8	เลือกดาวน์โหลด wxPython runtime for Python 2.4 . . . . .	72
ข.9	เลือกสถานที่ดาวน์โหลด . . . . .	72
ข.10	ขั้นตอนที่ 1 : ดับเบิลคลิกที่ไฟล์ติดตั้ง . . . . .	73
ข.11	ขั้นตอนที่ 2 : หน้าต้อนรับการติดตั้งให้ กด Next . . . . .	73
ข.12	ขั้นตอนที่ 3 : หน้ายอมรับข้อตกลงให้ กด Yes . . . . .	73
ข.13	ขั้นตอนที่ 4 : หน้าเลือกสถานที่ติดตั้ง ให้เลือกตามที่โปรแกรมได้กำหนดไว้แต่แรก . . . . .	73
ข.14	ขั้นตอนที่ 5 : หน้าเลือก component ให้เลือกทั้งหมด แล้วกด Next . . . . .	74
ข.15	ขั้นตอนที่ 6 : เข้าสู่ขั้นตอนการติดตั้งและเมื่อเสร็จแล้วให้เลือก checkbox ทั้งหมดเพื่อให้โปรแกรมติดตั้งทำการดัดแปลงระบบเพิ่มเติมแล้วกด Finish . . . . .	74
ข.16	ดาวน์โหลด Stanís Python Editor Version 0.8.2.a สำหรับ Windows ได้จากลิงค์ SourceForge (mirror) . . . . .	75
ข.17	เลือกสถานที่ดาวน์โหลด . . . . .	75
ข.18	ขั้นตอนที่ 1 : ดับเบิลคลิกที่ไฟล์ติดตั้ง . . . . .	76
ข.19	ขั้นตอนที่ 2 : หน้าต้อนรับการติดตั้งให้ กด Next . . . . .	76
ข.20	ขั้นตอนที่ 3 : หน้าเลือกสถานที่ติดตั้ง ให้เลือกตามที่โปรแกรมได้กำหนดไว้แต่แรก . . . . .	76
ข.21	ขั้นตอนที่ 4 : เข้าสู่ขั้นตอนการติดตั้งโดยกด Next เพื่อเริ่มการติดตั้ง . . . . .	77
ข.22	ขั้นตอนที่ 5 : เข้าสู่ขั้นตอนการติดตั้ง และเสร็จสิ้นการติดตั้ง . . . . .	77
ค.1	เปิดโปรแกรม Stanís Python Editor หรือเรียกสั้น ๆ ว่า SPE ขึ้นมา . . . . .	79
ค.2	หน้าต่างโปรแกรม Stanís Python Editor (SPE) . . . . .	80
ค.3	Shell . . . . .	81
ค.4	Local object browser . . . . .	82
ค.5	Session . . . . .	82
ค.6	Output . . . . .	82

ง.1	สั่งให้โปรแกรมทำงาน . . . . .	85
ง.2	ผลการทำงาน . . . . .	85
ง.3	การแจ้ง Error เพื่อใช้ประกอบการ Debug . . . . .	86

# บทที่ 1

## แนะนำภาษาไพธอน

**ไพธอน (Python)** เป็นภาษาโปรแกรมในลักษณะภาษาอินเทอร์พรีเตอร์โปรแกรมมิ่ง (Interpreted programming language) ผู้คิดค้นคือ Guido van Rossum ในปี 1990 ซึ่งไพธอนเป็นการจัดการชนิดของตัวแปรแบบแปรผันตามข้อมูลที่บรรจุอยู่ (Fully dynamically typed) และใช้การจัดการหน่วยความจำเป็นอัตโนมัติ (Automatic memory management) โดยได้เป็นการพัฒนาและผสมผสานของภาษาอื่น ๆ ได้แก่ ABC, Modula-3, Icon, ANSI C, Perl, Lisp, Smalltalk และ Tcl และภาษาไพธอนยังเป็นแนวคิดที่ทำให้เกิดภาษาใหม่ ๆ ซึ่งได้แก่ Ruby และ Boo เป็นต้น

ไพธอนนั้นพัฒนาเป็นโครงการ Open source โดยมีการจัดการแบบไม่หวังผลกำไรโดย Python Software Foundation และสามารถหาข้อมูลและตัวแปลภาษาได้จากเว็บไซต์ของไพธอนเอง ที่ <http://www.python.org/> ซึ่งในปัจจุบัน (ณ.วันที่ 25 กันยายน 2549) ไพธอนได้พัฒนาถึงรุ่นที่ 2.5 (ออกวันที่ 19 กันยายน 2549)

\* เอกสารเล่มนี้ยึดตามไพธอนรุ่นที่ 2.4.3 (ออกวันที่ 29 มีนาคม 2549)

### 1.1 ประวัติ

#### 1.1.1 Python 1.0

ไพธอนสร้างขึ้นครั้งแรกในปี 1990 โดย Guido van Rossum ที่ CWI (National Research Institute for Mathematics and Computer Science) ในประเทศเนเธอร์แลนด์ โดยได้นำความสำเร็จของภาษาโปรแกรมมิ่งที่ชื่อ ABC มาปรับใช้กับ Modula-3, Icon, C, Perl, Lisp, Smalltalk และ Tcl โดย Guido van Rossum ถือว่าเป็นผู้ริเริ่มและคิดค้น แต่เขาก็ยังคิดว่าผลงานอย่างไพธอนนั้น เป็นผลงานความรู้ที่สร้างขึ้นเพื่อความสนุกสนานโดยได้อ้างอิงงานชิ้นนี้ของเขาว่าเป็น Benevolent Dictator for Life (BDFL) ซึ่งผลงานที่ถูกเรียกว่าเกิดจากความสนุกสนานเหล่านี้มันมักถูกเรียกว่า BDFL เพราะมักเกิดจากความไม่ตั้งใจและความอยากที่จะทำอะไรที่เป็นอิสระนั่นเอง ซึ่งคนที่ถูกกล่าวถึงว่าทำในลักษณะแบบนี้ก็ได้แก่ Linus Torvalds ผู้สร้าง Linux kernel, Larry Wall ผู้สร้าง Perl programming language และคนอื่น ๆ อีกมากมาย

โดยที่ในไพธอน 1.2 นั้นได้ถูกปล่อยออกมาในปี 1995 โดย Guido ได้กลับมาพัฒนาไพธอนต่อที่ Corporation for National Research Initiatives (CNRI) ที่ เรสตัน, มลรัฐเวอร์จิเนีย ประเทศสหรัฐอเมริกา โดยที่ในขณะเดียวกันก็ได้ปล่อยรุ่นใหม่ ในหมายเลขรุ่น 1.6 ออกมาโดยอยู่ที่ CNRI เช่นกัน

ซึ่งหลังจากปล่อยรุ่น 1.6 ออกมาแล้ว Guido van Rossum ก็ได้ออกจาก CNRI เพื่อทำงานให้การทำธุรกิจพัฒนาซอฟต์แวร์แบบเต็มตัว โดยก่อนที่จะเริ่มทำงานธุรกิจ เขาก็ได้ทำให้ไพธอนนั้นอยู่บนสัญญาสิทธิแบบ General Public License (GPL) โดยที่ CNRI และ Free Software Foundation (FSF) ได้รวมกันเปิดเผยรหัสโปรแกรมทั้งหมด เพื่อให้ไพธอนนั้นได้ชื่อว่าเป็นซอฟต์แวร์เสรี และเพื่อให้ตรงตามข้อกำหนด



รูปที่ 1.1: Guido van Rossum

ของ GPL-compatible ด้วย (แต่ยังคงไม่สมบูรณ์เพราะการพัฒนาในรุ่น 1.6 นั้นออกมาก่อนที่จะใช้สัญญาสิทธิแบบ GPL ทำให้ยังมีบางส่วนของโค้ดที่ยังเปิดเผยไม่ได้)

และในปีเดียวกันนั่นเอง Guido van Rossum ก็ได้รับรางวัลจาก FSF ในชื่อว่า "Advancement of Free Software"

โดยในปีนั้นเองไพธอน 1.6.1 ก็ได้ออกมาเพื่อแก้ปัญหาข้อผิดพลาดของตัวซอฟต์แวร์และให้เป็นไปตามข้อกำหนดของ GPL-compatible license อย่างสมบูรณ์

### 1.1.2 Python 2.0

ในปี 2000 Guido และ Python Core Development team ได้ย้ายการทำงานไป BeOpen.com โดยที่พวกเขาได้ย้ายจาก BeOpen PythonLabs team โดยในไพธอนรุ่นที่ 2.0 นั้นได้ถูกนำออกเผยแพร่ต่อบุคคลทั่วไปจากเว็บไซต์ BeOpen.com และหลังจากที่ไพธอนออกรุ่นที่ 2.0 ที่ BeOpen.com แล้ว Guido และนักพัฒนาคนอื่น ๆ ในทีม PythonLabs ก็ได้เข้าร่วมกับทีมงาน Digital Creations

ไพธอนรุ่น 2.1 ได้สืบทอดการทำงานและพัฒนามาจาก 1.6.1 มากกว่าไพธอนรุ่น 2.0 และได้ทำการเปลี่ยนชื่อสัญญาสิทธิใหม่เป็น Python Software Foundation License โดยที่ในไพธอนรุ่น 2.1 alpha นั้นก็ได้เริ่มชื่อสัญญาสิทธินี้และผู้เป็นเจ้าของคือ Python Software Foundation (PSF) โดยที่เป็นองค์กรที่ไม่หวังผลกำไรเช่นเดียวกับ Apache Software Foundation

### 1.1.3 อนาคต

ผู้พัฒนาไพธอนมีการประชุมและถกเถียงกันในเรื่องของความสามารถใหม่ ๆ ในไพธอนรุ่นที่ 3.0 โดยมีชื่อโครงการว่า Python 3000 (Py3K) โดยที่จะหยุดการสนับสนุนโค้ดโปรแกรมจากรุ่น 2.x โดยที่ทำแบบนี้เพื่อทำการปรับปรุงเปลี่ยนแปลงการทำงานของภาษาให้ดียิ่งขึ้นตามคำแนะนำที่ว่า "reduce feature duplication by removing old ways of doing things" (ลดทอนคุณสมบัติที่ซ้ำซ้อนด้วยการยกเลิกเส้นทางที่เดินผ่านมาแล้ว) โดยในตอนนี้นั้นยังไม่มีตารางงานของไพธอน รุ่น 3.0 แต่อย่างใด แต่ Python Enhancement Proposal (PEP) ได้มีการวางแผนไว้แล้ว โดยได้วางแผนไว้ดังนี้

- ทำการเพื่อส่วนสนับสนุนชนิดตัวแปรให้มากขึ้น

- สนับสนุนการทำงานของชนิดตัวแปรแบบ unicode/str และ separate mutable bytes type
- ยกเลิกการสนับสนุนคุณสมบัติของ classic class, classic division, string exceptions และ implicit relative imports
- ฯลฯ

## 1.2 หลักปรัชญาของภาษาไพธอน

ไพธอนเป็นภาษาที่สามารถสร้างงานได้หลากหลายกระบวนทัศน์ (Multi-paradigm language) โดยจะมองอะไรที่มากกว่าการ coding เพื่อนำมาใช้งานตามรูปแบบเดิม ๆ แต่จะเป็นการนำเอาหลักการของกระบวนทัศน์ (Paradigm) แบบ Object-oriented programming, Structured programming, Functional programming และ Aspect-oriented programming นำเอามาใช้ทั้งแบบเดียว ๆ และนำมาใช้ร่วมกัน ซึ่งไพธอนนั้นเป็น ภาษาที่มีการตรวจสอบชนิดตัวแปรแบบยืดหยุ่น (dynamically type-checked) และใช้ Garbage collection ในการจัดการหน่วยความจำ

## 1.3 Language Evaluation Criteria

ด้วยความที่ไพธอนนั้นผสมผสานการสร้างภาษาที่สวยงาม ทำให้การอ่านหรือเข้าใจได้ (Readability) ต่าง ๆ นั้นทำได้ง่าย รวมถึงการเขียนโค้ด (Writability) ที่กระชับและสั้นในการเขียน รวมถึงมีประสิทธิภาพ ทำให้มีเสถียรภาพ (Reliability) สูงขึ้นและมีความรวดเร็วในการทำงานอีกด้วย และในด้านค่าใช้จ่าย (Cost) ในการพัฒนาซอฟต์แวร์จากไพธอนนั้นในประเทศไทยนั้นยังต้องใช้ค่าใช้จ่ายค่อนข้างสูง เพื่อให้ได้มาซึ่งซอฟต์แวร์ที่ดี เพราะผู้เชี่ยวชาญที่เขียนไพธอนได้มีเสถียรภาพนั้นยังมีน้อย ทำให้ค่าตัวสำหรับผู้พัฒนานั้นสูงตามไปด้วย ถึงแม้ว่าเครื่องมือในการพัฒนานั้นจะฟรี และเป็น Open source ก็ตาม แต่ค่าใช้จ่ายในด้านบุคลากรนั้นมีมากกว่าค่าเครื่องมือพัฒนา

## 1.4 ข้อเด่นของภาษาไพธอน

1. ง่ายต่อการเรียนรู้ โดยภาษาไพธอนมีโครงสร้างของภาษาไม่ซับซ้อนเข้าใจง่าย ซึ่งโครงสร้างภาษาไพธอนจะคล้ายกับภาษาซีมาก เพราะภาษาไพธอน สร้างขึ้นมาโดยใช้ภาษาซี ทำให้ผู้ที่คุ้นเคยภาษาซี อยู่แล้วใช้งานภาษาไพธอนได้ไม่ยาก นอกจากนี้โดยตัวภาษาเองมีความยืดหยุ่นสูงทำให้การจัดการกับงานด้านข้อความ และ Text File ได้เป็นอย่างดี
2. ไม่ต้องเสียค่าใช้จ่ายใดๆ ทั้งสิ้น เพราะตัวแปรภาษาไพธอนอยู่ภายใต้ลิขสิทธิ์ Python Software Foundation License (PSFL) ซึ่งเป็นของ Python Software Foundation (PSF) ซึ่งมีลักษณะคล้ายกับลิขสิทธิ์แม่แบบอย่าง General Public License (GPL) ของ Free Software Foundation (FSF)
3. ใช้ได้หลายแพลตฟอร์ม ในช่วงแรกภาษาไพธอนถูกออกแบบใช้งานกับระบบ Unix อยู่ก็จริง แต่ในปัจจุบันได้มีการพัฒนาตัวแปลภาษาไพธอน ให้สามารถใช้กับระบบปฏิบัติการอื่นๆ อาทิเช่น Linux Platform, Windows Platform, OS/2, Amiga, Mac OS X และรวมไปถึงระบบปฏิบัติการที่ .NET Framework, Java virtual machine ทำงานได้ ซึ่งใน Nokia Series 60 ก็สามารทำงานได้เช่นกัน

4. ภาษาไพธอนถูกสร้างขึ้นโดยได้รวบรวมเอาส่วนดีของภาษาต่างๆ เข้ามาไว้ด้วยกัน อาทิเช่น ภาษา ABC, Modula-3, Icon, ANSI C, Perl, Lisp, Smalltalk และ Tcl
5. ไพธอนสามารถรวมการพัฒนาของระบบเข้ากับ COM, .NET และ CORBA objects
6. สำหรับ Java libraries แล้วสามารถใช้ Jython เพื่อทำการพัฒนาซอฟต์แวร์จากภาษาไพธอนสำหรับ Java Virtual Machine
7. สำหรับ .NET Platform แล้ว สามารถใช้ IronPython ซึ่งเป็นการพัฒนาของ Microsoft เพื่อจะทำให้ไพธอนนั้นสามารถทำงานได้บน .Net Framework ซึ่งใช้ชื่อว่า Python for .NET
8. ไพธอนนั้นสนับสนุน Internet Communications Engine (ICE) และการรวมกันของเทคโนโลยีอื่น ๆ อีกมากมายในอนาคต
9. บางครั้งนักพัฒนาอาจจะพบว่าไพธอนไม่สามารถทำงานบางอย่างได้ แต่นักพัฒนาต้องการให้มันทำงานได้ ก็สามารถพัฒนาเพิ่มได้ในรูปแบบของ extension modules ซึ่งอยู่ในรูปแบบของโค้ด C หรือ C++ หรือใช้ SWIG หรือ Boost.Python
10. ภาษาไพธอนเป็นสามารถพัฒนาเป็นภาษาประเภท Server side Script คือการทำงานของภาษาไพธอนจะทำงานด้านฝั่ง Server แล้วส่งผลลัพธ์กลับมาฝั่ง Client ทำให้มีความปลอดภัยสูง และยังใช้ภาษาไพธอนนำมาพัฒนาเว็บเซอร์วิสได้อีกด้วย
11. ใช้พัฒนาระบบบริหารการสร้างเว็บไซต์สำเร็จรูปที่เรียกว่า Content Management Systems (CMS) ซึ่ง CMS ที่มีชื่อเสียงมาก และเบื้องหลังทำงานด้วยไพธอนคือ Plone <http://www.plone.org/>

## 1.5 Category และ Application Domains

ภาษาไพธอนนั้น จัดอยู่ใน Category ภาษาที่สามารถสร้างงานได้หลากหลายกระบวนทัศน์ (Multi-paradigm language) โดยรองรับทั้ง Object-oriented programming, Imperative, Functional programming และ Logic programming ซึ่งไพธอนสามารถนำไปพัฒนาซอฟต์แวร์ประยุกต์ได้มากมาย ได้แก่

### 1.5.1 Web และ Internet Development

ไพธอนนั้นมีการสนับสนุนในด้านของ Web Development ในโซลูชันระดับสูงด้วย Zope, mega frameworks อย่าง Django และ TurboGears และรวมไปถึง Content Management Systems ชั้นสูงอย่าง Plone และ CPS จึงทำให้ไพธอนนั้นเป็น Common Gateway Interface (CGI) ระดับสูงที่มีประสิทธิภาพที่ดีที่สุดตัวหนึ่งในตลาด

### 1.5.2 Database Access

ไพธอนนั้นสนับสนุนการเข้าถึงข้อมูลในฐานข้อมูลของผู้ผลิตฐานข้อมูลต่าง ๆ มากมาย โดยผ่านทาง ODBC Interfaces และ Database Connection Interface อื่น ๆ ซึ่งสามารถทำงานร่วมกับ MySQL, Oracle, MS SQL Server, PostgreSQL, SybODBC และอื่น ๆ ที่จะมีการเพิ่มเติมอีกในอนาคต



### 1.5.3 Desktop GUI

เมื่อไพธอนได้ติดตั้งลงบนเครื่องของคุณแล้ว จะมี Tk GUI development library ซึ่งเป็น libraries ที่มีความสามารถเทียบเท่า Microsoft Foundation Classes (MFC, ซึ่งคล้าย ๆ กับ win32 extensions), wxWidgets, GTK, Qt, Delphi และอื่น ๆ ทำให้สามารถพัฒนาซอฟต์แวร์ประยุกต์ต่าง ๆ แบบ Graphic user interface ได้

### 1.5.4 Scientific และ Numeric computation

ไพธอนรองรับการทำงานของนักวิทยาศาสตร์ในเรื่องของทฤษฎีการคำนวณ, Bioinformatics และ Physics เป็นต้น

### 1.5.5 Education

ไพธอนนั้นเป็นภาษาที่เหมาะสมกับการเรียนการสอนในวิชา programming อย่างมาก โดยสามารถนำไปใช้ในระดับเบื้องต้นถึงระดับสูง ซึ่ง Python Software Foundation นั้นได้มีหลักสูตรสำหรับการเรียนการสอนในด้านนี้อยู่แล้ว ซึ่งสามารถนำเอา pyBiblio และ Software Carpentry Course มาเรียนเพื่อเสริมความรู้ได้

### 1.5.6 Network programming

เป็นการเพิ่มความมาารจาก Web และ Internet Development ไพธอนนั้นสนับสนุนในการเขียนโปรแกรมในระดับต่ำในด้านของ network programming ที่ง่ายต่อการพัฒนา sockets และ รวมไปถึงการทำงานร่วมกับ mudules อย่าง Twisted และ Framework สำหรับ Asynchronous network programming

### 1.5.7 Software builder และ Testing

ไพธอนนั้นสนับสนุนการพัฒนาซอฟต์แวร์ที่มีการควบคุมการพัฒนาและจัดการระบบทดสอบต่าง ๆ โดยใช้เครื่องมือในการพัฒนาที่สนับสนุนการเขียนโปรแกรมในไพธอนเอง ซึ่งตัวไพธอนนั้นได้มาพร้อมกับ

- Scons สำหรับ build โปรแกรม
- Buildbot และ Apache Gump ที่ใช้สำหรับงาน Automated continuous compilation และ Testing
- Roundup หรือ Trac สำหรับ bug tracking และ project management

### 1.5.8 Game และ 3D Graphics Rendering

ไพธอนนั้นได้ถูกใช้ในตลาดพัฒนาเกมส์ทั้งเชิงธุรกิจและสมัครเล่น โดยมีการสร้าง Framework สำหรับพัฒนา Game บนไพธอนซึ่งชื่อว่า PyGame และ PyKyras ซึ่งยังรวมไปถึงการทำ 3D Graphics Rendering ที่ไพธอนมี libraries ทางด้านงานนี้อยู่มากมาย

## 1.6 ซอฟต์แวร์ที่เขียนด้วยไพธอน

- **BitTorrent** เป็นการพัฒนาโดยระบบการจัดการไฟล์ BitTorrent, การจัดการ การกระจายตัวของ Package ข้อมูลใน Tracker และการเข้ารหัสส่วนข้อมูลต่าง ๆ

- **Blender** ซอฟต์แวร์ open source สำหรับทำ 3D modeling
- **Chandler** ซอฟต์แวร์จัดการข้อมูลส่วนบุคคล (Personal Information Manager, PIM) โดยมีส่วนเพิ่มเติมทั้งงานปฏิทิน, อีเมล, ตารางงาน และข้อมูลโน้ตต่าง ๆ ซึ่งทำงานคล้าย ๆ กับ Outlook ของ Microsoft
- **Civilization IV** วิดีโอเกมส์ และยังเป็นเกมส์ที่ใช้ boost.python เพื่อทำการควบคุมส่วนประกอบต่าง ๆ ภายในเกมส์ ซึ่งรวมไปถึงรูปแบบ, หน้าตา และเนื้อหาของเกมส์ด้วย
- **Mailman** หนึ่งในซอฟต์แวร์ E-Mail mailing lists ที่ได้รับความนิยมสูงสุด
- **Kombilo** ระบบจัดการฐานข้อมูลของเกมส์โกะ
- **MoinMoin** ระบบ Wiki ที่ได้รับความนิยมสูงตัวหนึ่ง
- **OpenRPG** ระบบเกมส์เสมือนแบบ Role Playing Games บน Internet
- **Plone** ระบบ Content Management System
- **Trac** ระบบติดตามติดตามข้อผิดพลาดและจัดการข้อมูลด้านการพัฒนาซอฟต์แวร์ด้วย MoinMoin ที่เป็น wiki และ Subversion เพื่อทำระบบ Source version control
- **Turbogears** ระบบพัฒนาซอฟต์แวร์ Framework โดยรวมเอา CherryPy, SQLAlchemy, MochiKit และ KID templates
- **ViewVC** ระบบ Web-based สำหรับจัดการด้าน CVS และ SVN repositories
- **Zope** ระบบพัฒนาซอฟต์แวร์บนอินเทอร์เน็ตแบบ web-application platform
- **Battlefield 2** เกมส์ First Person Shooter ที่ได้ใช้ไพธอนในการทำ Configuration scripts
- **Indian Ocean Tsunami Detector** ซอฟต์แวร์สำหรับมือถือเพื่อแจ้งเตือน Tsunami
- **EVE Online** เกมส์แบบ Multi Massive Online Role Playing Game ซึ่งเป็นเกมส์ที่ได้รับอันดับสูงมากบน MMORPG.com
- **SPE - Stani's Python Editor** เป็น Free และ open-source สำหรับงานพัฒนาซอฟต์แวร์ด้วยไพธอน โดยมีทั้งแบบ Python IDE for Windows, Linux & Mac with wxGlade (GUI designer), PyChecker (Code Doctor) และ Blender (3D)
- ฯลฯ

## 1.7 ตัวอย่างความสำเร็จของไพธอน

**Industrial Light & Magic** "ไพธอนเป็นกุญแจสำหรับการสร้างผลงานที่ดี ถ้าไม่มีมันแล้วงานอย่าง Star Wars: Episode II ก็เป็นเรื่องที่ยากมากที่จะสำเร็จ ด้วยวิธีการ crowd rendering เพื่อส่งไปทำการ batch processing ในการ compositing video นั้นเป็นเรื่องที่ง่ายไปเลยเมื่อใช้การพัฒนาระบบด้วยไพธอน" **Tommy Burnette, Senior Technical Director, Industrial Light & Magic**  
 "ไพธอนอยู่ทุก ๆ ที่ใน ILM มันช่วยให้เราสามารถที่จะทำงานกับภาพกราฟฟิกที่ถูกสร้างสรรค์ได้ง่าย

และรวดเร็ว"**Philip Peterson, Principal Engineer, Research & Development, Industrial Light & Magic**

**Google** "ไฟธอนมีความสำคัญต่อ Google มาก เพราะตั้งแต่เริ่มมี Google เราก็มั่นสร้างระบบของเรา และยังคงเป็นส่วนสำคัญจนทุกวันนี้ โดยในทุก ๆ วันเหล่าวิศวกรของ Google ใช้ไฟธอนในการทำงานอยู่ตลอดเวลา เพื่อค้นหาข้อมูลบนโลกของอินเทอร์เน็ตอย่างไม่มีที่สิ้นสุด" **Peter Norvig, Director of Search Quality, Google, Inc.**

**NASA** "NASA ใช้ไฟธอนในการพัฒนา การจัดการ Model, Integration และ ระบบ Transformation ในงาน CAD/CAE/PDM โดยพวกเราเลือกไฟธอนเพราะมีความสามารถในการสร้างงานให้ออกมาได้อย่างรวดเร็วและมีประสิทธิภาพสูง โดยสิ่งที่สำคัญคือ code ในการเขียนนั้นสะอาดและง่ายต่อการจัดการดูแลในภายหลัง อีกทั้งยังมี libraries ให้ใช้อย่างมากมายทำการ Integration ของระบบนั้นเป็นไปอย่างชาญฉลาดและรวดเร็วแถมยังทำระบบที่สามารถเชื่อมต่อกับระบบอื่น ๆ ได้อย่างดี ซึ่งไฟธอนนั้นตอบโจทย์ของเราได้ทั้งหมด" **Steve Waterbury, Software Group Leader, NASA STEP Testbed**



## บทที่ 2

### การแสดงผลเบื้องต้น (Printing)

การแสดงผลออกทางหน้าจอของไพธอนนั้นใช้คำสั่ง *print* แล้วตามด้วย 'String' หรือ "String" โดยแทนที่ String ด้วยข้อความใด ๆ เช่น

\* ในภาษา C/C++, Java, ฯลฯ เครื่องหมายที่บอกการจบ ของ คำสั่งคือ ; (Semi-colon Symbol) แต่ Python ใช้การจบบรรทัดแทน

```
>>> print "Hello, World!"
```

และได้ผลการทำงาน

```
Hello, World!
```

ทดสอบการพิมพ์หลาย ๆ บรรทัด

```
>>> print "Jack and Jill went up a hill"
>>> print "to fetch a pail of water;"
>>> print "Jack fell down, and broke his crown,"
>>> print "and Jill came tumbling after."
```

ผลการทำงาน

```
Jack and Jill went up a hill
to fetch a pail of water;
Jack fell down, and broke his crown,
and Jill came tumbling after.
```

ซึ่งบางครั้งเราต้องการพิมพ์ชุดข้อความซ้ำ ๆ กันเราสามารถใส่สัญลักษณ์ \* (Repetition Symbol) เพื่อทำการพิมพ์ชุดข้อความนั้นได้

```
>>> print "Hello, World! "*5
```

ผลการทำงาน

```
Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!
```

\* ภาษาไพธอน ใช้ "#" บอกจุดเริ่มต้นของ Comment ไปจนสุดบรรทัด โดย Comment เขียนไว้เตือนความจำ ไม่ใช่ส่วนที่เอาไป Execute Program  
ตัวอย่าง

```
>>> print "Foo"  
# Test Comment  
>>> print "Bar"
```

ผลการทำงาน

```
Foo  
Bar
```

## บทที่ 3

# การตั้งชื่อแปร และคำสงวน (Reserved word หรือ Keywords)

### 3.1 การตั้งชื่อแปร

1. ขึ้นต้นด้วยตัวอักษรในภาษาอังกฤษ ตามด้วยตัวอักษรหรือตัวเลขใดๆ ก็ได้
2. ห้ามเว้นช่องว่าง และห้ามใช้สัญลักษณ์พิเศษนอกจาก underscore (\_) เท่านั้น
3. ตัวอักษรของชื่อจะคำนึงถึงความแตกต่างระหว่างอักษรตัวพิมพ์ใหญ่กับตัวพิมพ์เล็ก
4. การตั้งชื่อมีชื่อฟังก์ชันว่าจะต้องไม่ซ้ำกับคำสงวน(Reserved words, Keywords)
5. ควรจะตั้งชื่อโดยให้ชื่อนั้นมีสื่อความหมายให้เข้ากับข้อมูล สามารถอ่านและเข้าใจได้

### 3.2 คำสงวนในการใช้งาน (Reserved words, Keywords)

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, as (ไพธอน 2.5) และ with (ไพธอน 2.5)

\* อ้างอิงจากเอกสารไพธอน รุ่น 2.4.3 และ 2.5 (Python Reference Manual Release 2.4.3, documentation updated on 29 March 2006 and Python Reference Manual Release 2.5, documentation updated on 19 September 2006)





# บทที่ 4

## การคำนวณทางคณิตศาสตร์ (Arithmetic Mathematics)

### 4.1 การคำนวณพื้นฐาน (normal arithmetic operators)

ไพธอนนั้นมีสัญลักษณ์การคำนวณพื้นฐาน (normal arithmetic operators) ซึ่งได้แก่

- \*\* (ยกกำลัง, Exponentiation)
- \* (คูณ, multiplication)
- / (หาร, division)
- % (หารเอาเศษ, remainder หรือ modulo)
- + (บวก, addition)
- - (ลบ, subtraction)

โดยอันดับความสำคัญของการคำนวณเหมือนกับคณิตศาสตร์โดยมีความสำคัญดังต่อไปนี้

1. วงเล็บ (parentheses "()")
2. ยกกำลัง (exponents "\*\*" )
3. คูณ (multiplication "\*" ), หาร (division "/" ) และหารเอาเศษ (remainder/modulo "%" )
4. บวก (addition "+") และ ลบ (subtraction "-")

ทดสอบการคำนวณได้จากการคำนวณด้านล่างนี้

```
>>> i = 10
>>> f = 6.54
>>> print i + f
16.54
```

อธิบายโปรแกรมด้านบนได้ว่า

ตัวแปร  $i$  ถูกตั้งค่าเริ่มต้นไว้ที่ 10 และตัวแปร  $f$  ถูกตั้งค่าเริ่มต้นเป็น 6.54 โดยที่  $i$  มีชนิดตัวเป็น integer และ  $f$  เป็น float โดยเรานำมาบวกกันจะได้คำตอบคือ 16.54

ทดสอบการบวกโดยให้นำค่าที่ทำการคำนวณส่งกลับตัวแปรเดิม จากคำสั่งการทำงานด้านล่างนี้

```
>>> i = 10
>>> i = i + 6
>>> i = i / 2
>>> print i
8
```

จะเห็นว่าชุดคำสั่งที่ใช้มีความยาวกว่าปกติ แต่อ่านง่ายกว่า ซึ่งเราสามารถย่อรูปการคำนวณต่าง ๆ เหล่านี้ ได้ดังตัวอย่างด้านล่าง

```
>>> i = 10
>>> i += 6
>>> i /= 2
>>> print i
8
```

ซึ่งการย่อรูปแบบนี้เป็นไปตามรูปแบบดั้งเดิมของภาษา C นั่นเอง

## 4.2 การคำนวณผ่านฟังก์ชันภายใน (Built-in Math Functions)

### 4.2.1 การหาค่าสัมบูรณ์ (absolute value)

มีรูปแบบ Function คือ  $abs(var)$  เป็น function ที่ใช้หาค่าสัมบูรณ์ โดนที่ค่า  $var$  เป็นตัวแปรหรือจำนวนที่ต้องการหาค่า

```
>>> print abs(-6.5)
6.5
```

### 4.2.2 จำนวนที่น้อยที่สุด และมากที่สุดในกลุ่ม (smallest or largest values)

ฟังก์ชันหาจำนวนที่น้อยที่สุด มีรูปแบบฟังก์ชัน คือ  $min(var)$  เป็นฟังก์ชันที่ใช้ในการหาจำนวนที่น้อยที่สุดในกลุ่มของชุดข้อมูลชนิดต่าง ๆ เช่น list, set หรือแม้แต่ตัวเลขทั่วไป โดยที่ค่า  $var$  เป็นตัวแปร, จำนวน หรือชุดของจำนวนที่ต้องการหาค่า

ฟังก์ชันหาจำนวนที่มากที่สุด มีรูปแบบฟังก์ชัน คือ  $max(var)$  เป็นฟังก์ชันที่ใช้ในการหาจำนวนที่มากที่สุดในกลุ่มของชุดข้อมูลชนิดต่าง ๆ เช่น list, set หรือแม้แต่ตัวเลขทั่วไป โดยที่ค่า  $var$  เป็นตัวแปร, จำนวน หรือชุดของจำนวนที่ต้องการหาค่า

```
>>> print min(6, 7, 2, 8, 5)
2
>>> print max(6, 7, 2, 8, 5)
```

```

8
>>> print min([0, 43.5, 19, 5, -6])
0
>>> print max([0, 43.5, 19, 5, -6])
43.5

```

### 4.2.3 กำหนดจำนวนตัวเลขทศนิยม (specified number of digits)

มีรูปแบบฟังก์ชัน คือ *round(var, digits)* เป็นฟังก์ชันที่ใช้ในการเพิ่มหรือลดจำนวนตัวเลขทศนิยมที่จะนำมาแสดง โดย

```

>>> print round(1234.56789, 2)
1234.57
>>> print round(1234.56789, -2)
1200.0

```

### 4.2.4 หาผลรวมทั้งหมดในชุดข้อมูล (adds numbers in a sequence.)

มีรูปแบบฟังก์ชัน คือ *sum(sequence)* เป็นฟังก์ชันที่ใช้ในการหาผลรวมของชุดข้อมูลตัวเลขหนึ่ง ๆ โดยที่ใส่ค่าของชุดจำนวนเข้าไปในรูปแบบของ sequence of numbers เช่น (1,2,3,4,5) ซึ่งหมายถึงจำนวนตั้งแต่ 1 - 5 เป็นต้น ลงไปในฟังก์ชัน sum เพื่อทำการคำนวณหาผลรวมของชุดข้อมูลดังกล่าว

```

>>> print sum((1, 2, 3, 4, 5))
15

```

### 4.2.5 ช่วงของข้อมูลตัวเลข (range of numbers.)

มีรูปแบบฟังก์ชัน คือ *range(start, end [,step])* เป็นฟังก์ชันที่ใช้ในการส่งค่าช่วงของข้อมูลตัวเลขที่ต้องการออกมาเช่นต้องการตัวเลขตั้งแต่ 1 - 500 เราสามารถทำได้ตามตัวอย่างด้านล่างนี้

```

>>> print range(1, 6)
[1, 2, 3, 4, 5]

```

ซึ่งเราสามารถนำความสามารถนี้มาใช้ร่วมกับฟังก์ชันหาผลรวมทั้งหมดได้ โดยดั่งตัวอย่างด้านล่างนี้

```

>>> print sum(range(1, 6))
15

```

จากตัวอย่างทั้งสองนั้น โดยช่วงของข้อมูลนั้นคือ 1 - 6 ซึ่งได้ข้อมูลคือ 1, 2, 3, 4, 5 ซึ่งเรานำมาซึ่งสูตรคือ  $m$  ถึง  $(n - 1)$  โดยที่  $m$  และ  $n$  คือจำนวนเต็มบวก และ  $m$  มีค่าน้อยกว่า  $n$  หรือบางครั้งเราอยากเพิ่มค่าทีละ 2 ก็สามารถทำได้ด้วย

```

>>> print range(1, 6, 2)
[1, 3, 5]

```



# บทที่ 5

## ชนิดของตัวแปร (Data type)

ดังที่เราได้ทราบไปแล้วว่าภาษาไพธอนเป็นภาษาที่เป็น Interpreter Programming ซึ่งเราไม่จำเป็นต้องสนใจ Data type แต่บางครั้งการที่เรารู้จัก Data type ต่าง ๆ และนำมาใช้งานได้อย่างเหมาะสมทำให้การเขียนโปรแกรมมีประสิทธิภาพมากขึ้น โดย Data type ต่าง ๆ นั้นมีจุดเด่นในแบบชนิดของตัวเอง

ซึ่งในการแสดงผลการทำงานของข้อมูลต่าง ๆ ในไพธอนนั้นมีหลากหลายรูปแบบ เราจะมาพูดถึงในเรื่องนี้เช่นเดียวกัน

โดยในหลาย ๆ ภาษานั้น ๆ เราจำเป็นต้องประกาศชนิดตัวแปร (Data type) ก่อน แล้วจึงตั้งชื่อตัวแปร ซึ่งหลาย ๆ ครั้งสร้างความสับสนในการจดจำ แต่ในภาษาไพธอนนั้นเราไม่จำเป็นต้องประกาศชนิดของตัวแปรก่อนการใช้งานแต่อย่างใด โดยในบทนี้เราจะพูดถึงชนิดของตัวแปรต่าง ๆ

### 5.1 ตัวเลข (Numbers)

#### 5.1.1 จำนวนเต็ม(Integers)

##### จำนวนเต็มธรรมดา (Plain Integer)

**Plain Integers** มีคำย่อสำหรับเขียนในโปรแกรมคือ *int* เป็นการบ่งบอกคุณสมบัติของตัวแปรที่ใช้เก็บข้อมูลตัวเลขแบบจำนวนเต็มที่เป็นตัวเลขแบบ signed integer เก็บข้อมูล 32 bits ตั้งแต่ -2147483648 ถึง +2147483647

ทดสอบโดยการพิมพ์คำสั่งโปรแกรมดังนี้

```
>>> x = 42
>>> type(x)
```

จะได้ผลการทำงานโดยแสดงเป็นชนิดของตัวแปรนั้นคือ

```
<type 'int'>
```

\* function *type(var)* เป็น function ที่ใช้ในการแสดงชนิดของตัวแปรนั้น ๆ โดยที่ตัวแปร *var* เป็นชื่อตัวแปรที่ต้องการแสดงชนิดของตัวแปรนั้น ๆ ดังตัวอย่างที่ได้ทำด้านบน

การทดสอบต่อมาเราจะทำการ casting data ผ่าน function *int* เพื่อทำการแปลงข้อมูลเป็น interger numbers ตัวอย่างเช่น



```
>>> debt = 7784834892156.63
>>> print debt
>>> type(debt)
```

ภาษาไพธอนนั้นจะทำการย่อขนาดของจำนวนให้อยู่ในรูปแบบขนาดย่อทางคณิตศาสตร์จะได้ผลการทำงานโดยแสดงเป็นชนิดของตัวแปรนั้นคือ

```
7.78483489216e+012
<type 'float'>
```

โดยมีความหมายว่า  $7.78483489216 \times 10^{12}$

เราสามารถทำให้จำนวนชนิด ๆ เปลี่ยนเป็นจำนวนจริงได้โดยใช้ function **float**

```
>>> x = float(17)
>>> y = float("4")
>>> print x, y, x - y
```

ผลการทำงาน

```
17.0 4.0 13.0
```

\* function **float**(var) เป็น function ที่ใช้ในการแปลงชนิดของข้อมูลของตัวแปรอื่น ๆ มาเป็นชนิดตัวแปรแบบ floating-point โดยที่ตัวแปร var เป็นชื่อตัวแปรที่ต้องการแปลงชนิดข้อมูลของตัวแปรนั้น ๆ

### 5.1.3 จำนวนเชิงซ้อน (Complex Numbers)

มีคำย่อสำหรับเขียนในโปรแกรมคือ *complex* จำนวนเชิงซ้อนคือเซตที่ต่อเติมจากเซตของจำนวนจริงโดยเพิ่มจำนวน  $j$  เข้าไปในตัวเลขจำนวนจริงจนได้เป็น จำนวนจินตภาพ (imaginary number) จนทำให้สมการ  $j^2 + 1 = 0$  เป็นจริง และหลังจากนั้นเพิ่มสมาชิกตัวอื่นๆ เข้าไปจนกระทั่งเซตที่ได้ใหม่มีสมบัติปิดภายใต้การบวกและการคูณ จำนวนเชิงซ้อน  $z$  ทุกตัวสามารถเขียนอยู่ในรูป  $x + iy$  โดยที่  $x$  และ  $y$  เป็นจำนวนจริง โดยเราเรียก  $x$  และ  $y$  ว่าส่วนจริงและส่วนจินตภาพของ  $z$  ตามลำดับ

เราสามารถนำมาเขียนเป็นสมการในการเขียนชุดคำสั่งได้ดังนี้

```
>>> imaginary_number = 16j
>>> complex_number = 6 + 4j
>>> print complex_number
(6+4j)
>>> print type(complex_number)
<type 'complex'>
```

หรือเราอาจจะไม่ต้องใช้จำนวน  $j$  เพื่อทำให้ตัวเลขนั้นเป็นจำนวนจินตภาพ ได้จาก function ที่ชื่อว่า **complex**(real, [imag]) ดังตัวอย่างด้านล่างนี้

\* function **complex**(real, imag) เป็น function ที่ใช้ในการแปลงชนิดของข้อมูลของตัวแปรอื่น ๆ มาเป็นชนิดตัวแปรแบบ complex โดยที่ตัวแปร real เป็นจำนวนที่เป็นจำนวนจริง และ imag คือจำนวนจินตภาพ

```
>>>complex_number = complex(6, 4)
>>>type(complex_number)
<type 'complex'>
>>>print(complex_number)
(6+4j)
```

## 5.2 ชนิดข้อมูลแบบการรวมกลุ่มข้อมูล (Collection Data Types)

### 5.2.1 ลิสต์ (List)

ในไพธอนนั้นชนิดตัวแปรที่ถูกนำมารวมกันอยู่ในชื่อเดียวกับเลยคือ อาร์เรย์ (Array) และ ลิสต์ (List) แต่ถ้านักพัฒนาต้องการใช้ อาร์เรย์แบบที่คุ้นเคยจริง ๆ จำเป็นต้อง import module array ของไพธอนเข้ามา ซึ่งยุ่งยากและทำงานช้ากว่าที่ควรจะเป็น

ลิสต์มีค่าย่อสำหรับเขียนในโปรแกรมคือ *list* การทำงานของลิสต์นั้นเป็นการนำข้อมูลหลาย ๆ ชนิดมาเรียงต่อกันในลิสต์ของตัวแปรนั้น ๆ ซึ่งแตกต่างจากอาร์เรย์ที่ข้อมูลที่นำมาเรียงต่อกันต้องเป็นชนิดเดียวกัน และนี่คงเป็นสาเหตุที่ทำให้อาร์เรย์ ถูกตัดทิ้งไปจากชนิดของตัวแปรพื้นฐานของไพธอนเพราะด้วยเหตุผลด้านความยืดหยุ่นของชนิดของตัวแปรนั่นเอง โดยลิสต์นั้นจะมีข้อมูลเรียงกันหลาย ๆ ตัว ครอบด้วยเครื่องหมาย square brackets "[" และ "]" เช่น

```
>>> i = [1, 2, 3, 4, 'Foo', 5, 'Bar']
>>> print i
>>> type(i)
```

ผลที่ได้คือ

```
[1, 2, 3, 4, 'Foo', 5, 'Bar']
<type 'list'>
```

เราสามารถเข้าถึงข้อมูลในแต่ละลิสต์ได้ ผ่านทางหมายเลขสมาชิก (index key) เช่น

```
>>> print i[6]
```

ผลการทำงานคือ

```
Bar
```

โดยหมายเลขสมาชิกเริ่มตั้งแต่ 0 ไปจนถึงจำนวนข้อมูลที่มีอยู่ในลิสต์ลบด้วยหนึ่ง หรืออธิบายให้เห็นภาพก็คือ

```
0, 1, 2, 3, 4 , . . . . , n - 1
```

เมื่อ  $n$  คือจำนวนของข้อมูลในลิสต์

เราสามารถเข้าถึงสมาชิกของ list เป็นกลุ่มๆ ได้ โดยใช้คำสั่ง : (Colon Symbol) เพื่อคั่นระหว่างหมายเลขสมาชิกที่ต้องการ เช่น

```
>>> x = [1, 2, 3, 4, 'Foo', 5, 'Bar']
>>> print x[3:5]
```



ผลการทำงาน

```
[4, 'Foo']
```

เราสามารถเลือกสมาชิกบางตัวอย่างมีเงื่อนไขได้ เช่น

```
>>> y = [10, 3, 5, 25, 7, 9]
>>> z = [x for x in y if x >= 9]
>>> print z
```

ผลการทำงานคือ

```
[10, 25, 9]
```

จากชุดคำสั่งด้านบนนั้นเป็นการเลือกสมาชิกภายในลิสต์ที่มีค่ามากกว่าหรือเท่ากับ 9 นั่นเอง

## 5.2.2 ดิกชันนารี (Dictionary หรือ Groupings of Data Indexed by Name)

ดิกชันนารี (Dictionary หรือ Groupings of Data Indexed by Name) มีคำย่อสำหรับเขียนในโปรแกรมคือ *dict* เป็นอนุกรมอีกอันหนึ่ง มีสภาพเหมือนอาเรย์ ที่มีสมาชิกทั้ง keys และ value โดยที่ นั้นจะใช้ชื่ออ้างอิงสมาชิก (associated key) แทนการใช้หมายเลขสมาชิก (index key) ซึ่งจะซ้ำกันไม่ได้ (ถ้ากำหนดค่าซ้ำ มันจะลบค่าเก่า และใช้ค่าใหม่แทน) โดยใช้การจัดเรียงข้อมูลและแก้ไขค่าไม่ได้ (จึงสามารถใช้ tuple เป็น keys ได้ ถ้าชนิดของข้อมูลสมาชิกเป็นชนิดเดียวกัน แต่ใช้ list เป็น keys ไม่ได้) แต่ลบ keys:value ได้ เวลาเรียกข้อมูล value จะค้นจาก keys ถ้าค้นไม่พบจะเกิดข้อผิดพลาด

```
>>> i = {'first': 'alpha', 'last': 'omega'}
>>> print i
>>> print i['first']
>>> type(i)
```

ผลการทำงานคือ

```
{'last': 'omega', 'first': 'alpha'}
'alpha'
<type 'dict'>
```

โดยการเข้าถึงข้อมูลของดิกชันนารี นั้นสามารถทำได้โดยการอ้างอิงจากชื่ออ้างอิงสมาชิก ดังในตัวอย่างข้างต้นนั้นได้อ้างอิงถึง *first* ด้วยรูปแบบ

```
>>> print i['first']
```

เราก็จะได้ข้อมูล *'alpha'* ออกมา

โดยเราสามารถใส่สมาชิกลงไปได้เรื่อย ๆ จากตัวอย่างต่อไปนี้

```
>>> menus_specials = {}
>>> menus_specials["breakfast"] = "canadian ham"
>>> menus_specials["lunch"] = "tuna surprise"
>>> menus_specials["dinner"] = "Cheeseburger Deluxe"
>>> print menus_specials['breakfast']
```

ผลการทำงาน

```
canadian ham
```

### 5.2.3 ทับเบิล (Tuples) และ อนุกรม (Sequences)

ทับเบิล (น้องลิสต์, Tuples) มีคำย่อสำหรับเขียนในโปรแกรมคือ *tuple* ข้อมูลจะอยู่ภายในวงเล็บ () แต่ตอนกำหนดค่า ถ้าอยู่โดดๆ อาจไม่ใส่วงเล็บก็ได้ โดยใช้หลักคล้ายกับลิสต์ (แต่ไม่เหมือนกันทั้งหมด) เพียงแต่เราสามารถนำลิสต์หลาย ๆ ลิสต์มาบรรจุลงในลิสต์เดียวกันเองได้โดยไม่ต้องแยกตัวแปร เช่น

```
>>> t = 12345, 54321, 'hello!'
>>> type(t)
<type 'tuple'>
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

แต่สมาชิกในทับเบิลเปลี่ยนค่าไม่ได้ ซึ่งเหมือนกับเหมือนสตริง ไม่เหมือนลิสต์ ซึ่งการกำหนดค่าให้ทับเบิลที่มีสมาชิกเดียว ยุ่งยากขึ้นเล็กน้อย เพราะต้องใช้ "," ช่วย

```
>>> empty = ()
>>> singleton = 'hello', # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

การกำหนดค่าให้ tuple เช่น `t = 12345, 54321, 'hello!'` เรียกว่าการแพ็ค (packing) ใช้ได้กับ tuple อย่างเดียว ส่วนการกำหนดค่าแบบย้อนกลับ เรียกว่าการ อั้นแพ็ค (unpacking) อั้นนี้ใช้ได้กับทุกอนุกรม คือ ทั้งลิสต์ และทูเปิล

\* function *len(var)* เป็น function ที่ใช้ในการหน้าจำนวนสมาชิก, ความยาวของสายอักขระ และกลุ่มของข้อมูลต่าง ๆ โดยที่ *var* นั้นคือชื่อของตัวแปรที่เราต้องการหาตัวเอง

```
>>> x, y, z = t
```

\* ในตัวอย่างนี้ `t` ต้องมีสมาชิกอย่างน้อย 3 ค่า มิเช่นนั้นจะเกิดข้อผิดพลาด เนื่องจากไม่สามารถให้ค่าได้กับ 3 ตัวแปรที่ต้องการส่งค่าไปให้

### 5.2.4 เซ็ต (Sets)

เซ็ต เปรียบเสมือนส่วนขยายของลิสต์(และสตริงด้วย) และจะไม่มีสมาชิกที่มีค่าซ้ำกัน ใช้ประโยชน์เหมือนกับ เรื่องเซตในวิชาคณิตศาสตร์

มีคำย่อสำหรับเขียนในโปรแกรมคือ *set* แต่การที่จะได้ชนิดข้อมูลแบบเซตนั้นต้องทำผ่าน function *set* เพื่อให้ได้เซตออกมา ซึ่งส่วนมากแล้วจะใช้ *List* มาทำการแปลงชนิดตัวแปรเป็นเซตอีกที ดังตัวอย่างด้านล่าง

\* function *set(var)* เป็น function ที่ใช้ในการแปลงชนิดของข้อมูลของตัวแปรอื่น ๆ มาเป็นชนิดตัวแปรแบบ *set* โดยที่ตัวแปร *var* เป็นชื่อตัวแปรที่ต้องการแปลงชนิดข้อมูลของตัวแปรนั้น ๆ

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)           # create a set without duplicates
>>> type(fruit)
<type 'set'>
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit           # fast membership testing
True
>>> 'crabgrass' in fruit
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                               # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                             # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                             # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                             # letters in both a and b
set(['a', 'c'])
>>> a ^ b                             # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

### 5.2.5 ฟังก์ชันที่น่าสนใจเกี่ยวข้องกับลิสต์และดิกชันนารี

\* การเรียกใช้ฟังก์ชันของลิสต์และดิกชันนารีนั้นทำคล้าย ๆ กับการเรียกใช้เมธอดในภาษา *Java* เช่น

```
>>> i = [1,2,3,4]
>>> print i.pop()
4
>>> j = {'first':'alpha','last':'omega'}
>>> print j.pop('first')
alpha
```

### ฟังก์ชัน pop - [ลิสต์/ดิกชันนารี]

รูปแบบฟังก์ชัน `Object.pop([key])` ในลิสต์นั้นเราไม่จำเป็นต้องกำหนดหมายเลขสมาชิก (index key) ก่อนการ pop เพราะเมธอดจะนำค่าบนสุดมาให้เรา แต่ถ้าต้องการ pop ในหมายเลขสมาชิกที่ต้องการก็เพียงแต่ใส่หมายเลขสมาชิกเท่านั้น แต่ในดิกชันนารีเราจำเป็นต้องกำหนดชื่ออ้างอิงสมาชิก (associated key) ก่อนการ pop เพราะมีเช่นนั้นเมธอดจะไม่สามารถนำค่าของสมาชิกนั้น ๆ มาให้เราได้

```
>>> i = [1,2,3,4]
>>> print i.pop()
4
>>> j = {'first':'alpha','last':'omega'}
>>> print j.pop('first')
alpha
```

### ฟังก์ชัน append - [ลิสต์]

รูปแบบฟังก์ชัน `Object.append([object])` เป็นฟังก์ชันที่ใช้ในการเพิ่มข้อมูลสมาชิกลงไปในลิสต์โดยจะนำไปต่อท้ายลิสต์เสมอ

```
>>> i = [1,2,3,4]
>>> print i.pop()
4
>>> i.append(5)
>>> print i
[1,2,3,5]
```

### ฟังก์ชัน insert - [ลิสต์]

รูปแบบฟังก์ชัน `Object.insert(index, object)` เป็นฟังก์ชันที่ใช้ในการเพิ่มข้อมูลสมาชิกลงไปในลิสต์โดยจะนำไปใส่ในลำดับสมาชิกที่เราต้องการ

```
>>> print i
[1, 3]
>>> i.insert(4, 5)
>>> print i
[1, 3, 5]
>>> i.insert(2, 6)
>>> print i
[1, 3, 6, 5]
```

### ฟังก์ชัน count - [ลิสต์]

รูปแบบฟังก์ชัน `Object.count([object])` เป็นฟังก์ชันที่ใช้ในการนับข้อมูลสมาชิกที่ต้องการค้นหา

```
>>> i = [1,2,3,4]
>>> print i.count(5)
```

```
0
>>> print i.count(4)
1
```

### ฟังก์ชัน index - [ลิสต์]

รูปแบบฟังก์ชัน *Object.index(value, [ start , stop ])* เป็นฟังก์ชันที่ใช้ในการหาค่าของหมายเลขสมาชิก (index key) ว่าข้อมูลที่เราต้องการนั้นอยู่ที่หมายเลขสมาชิกที่เท่าใด

```
>>> i = [1,2,3,4]
>>> print i.index(4)
3
>>> print i.index(1)
0
```

### ฟังก์ชัน extend - [ลิสต์]

รูปแบบฟังก์ชัน *Object.extend(list|dict)* เป็นฟังก์ชันที่ใช้ในเพิ่มสมาชิกทั้งหมดจากอีกลิสต์หรือดิกชันนารี จากตัวแปรอื่น ๆ มาไว้ที่ตัวแปรตั้งต้น

```
>>> j = {'last': 'omega'}
>>> i = [1,2,3,4]
>>> i.extend(j)
>>> print i
[1, 2, 3, 4, 'last']
```

### ฟังก์ชัน remove - [ลิสต์]

รูปแบบฟังก์ชัน *Object.remove(value)* เป็นฟังก์ชันที่ใช้ในการลบค่าที่กำหนดไว้ในพารามิเตอร์ออกไปจากลิสต์

```
>>> print i
[1, 2, 3, 4, 'last']
>>> i.remove(1)
>>> print i
[3, 6, 5, 'last']
```

### ฟังก์ชัน sort - [ลิสต์]

รูปแบบฟังก์ชัน *Object.sort()* เป็นฟังก์ชันที่ใช้ในการจัดเรียงข้อมูล (sort) ภายในลิสต์โดยสามารถที่จะจัดเรียงได้ทั้งตามหมายเลขสมาชิก หรือจัดเรียงตามข้อมูลของสมาชิก

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> print a
[1, 2, 3, 4, 5]
```

```
>>> i = ['a', 'r', 'b', 'i', 'z']
>>> print i
['a', 'r', 'b', 'i', 'z']
>>> i.sort()
>>> print i
['a', 'b', 'i', 'r', 'z']
```

### ฟังก์ชัน reverse - [ลิสต์]

รูปแบบฟังก์ชัน *Object.reverse()* เป็นฟังก์ชันที่ใช้ในการจัดเรียงข้อมูล (sort) ภายในลิสต์แบบย้อนกลับ

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> print a
[1, 2, 3, 4, 5]
>>> i.reverse()
>>> print i
[5, 4, 3, 2, 1]
```

### ฟังก์ชัน clear - [ดิกชันนารี]

รูปแบบฟังก์ชัน *Object.clear()* เป็นฟังก์ชันที่ใช้ในการลบข้อมูลภายในดิกชันนารีทั้งหมด

```
>>> j = {'first': 'a', 'second': 'b'}
>>> print j
{'second': 'b', 'first': 'a'}
>>> j.clear()
>>> print j
{}
```

### ฟังก์ชัน get - [ดิกชันนารี]

รูปแบบฟังก์ชัน *Object.get(key)* เป็นฟังก์ชันที่ใช้ในเรียกข้อมูลจากชื่ออ้างอิงสมาชิกภายในดิกชันนารี

```
>>> j = {'first': 'a', 'second': 'b'}
>>> print j
{'second': 'b', 'first': 'a'}
>>> j.get('first')
'a'
```

### ฟังก์ชัน has\_key - [ดิกชันนารี]

รูปแบบฟังก์ชัน *Object.has\_key(key)* เป็นฟังก์ชันที่ใช้ในการตรวจสอบชื่ออ้างอิงสมาชิก (associated key) ว่ามีอยู่หรือไม่

```
>>> j = {'first': 'a', 'second': 'b'}
>>> print j
{'second': 'b', 'first': 'a'}
>>> j.has_key('first')
True
>>> j.has_key('last')
False
```

### ฟังก์ชัน items - [ดิกชันนารี]

รูปแบบฟังก์ชัน *Object.items(key)* เป็นฟังก์ชันที่ใช้ในการแสดงรายการชื่ออ้างอิงสมาชิกและข้อมูลทั้งหมดในดิกชันนารี

```
>>> j = {'first': 'a', 'second': 'b'}
>>> print j
{'second': 'b', 'first': 'a'}
>>>>> j.items()
[('second', 'b'), ('first', 'a')]
```

### ฟังก์ชัน keys - [ดิกชันนารี]

รูปแบบฟังก์ชัน *Object.keys()* เป็นฟังก์ชันที่ใช้ในการแสดงรายการชื่ออ้างอิงสมาชิกทั้งหมดในดิกชันนารี

```
>>> j = {'first': 'a', 'second': 'b'}
>>> print j
{'second': 'b', 'first': 'a'}
>>>>> j.keys()
['second', 'first']
```

### ฟังก์ชัน values - [ดิกชันนารี]

รูปแบบฟังก์ชัน *Object.values()* เป็นฟังก์ชันที่ใช้ในการแสดงรายการข้อมูลทั้งหมดในดิกชันนารี

```
>>> j = {'first': 'a', 'second': 'b'}
>>> print j
{'second': 'b', 'first': 'a'}
>>>>>>>> j.values()
['b', 'a']
```

## 5.3 สายอักขระ (String หรือ Array of Characters)

สายอักขระ (Strings) เป็นการเรียงตัวของอักขระมาต่อกันมากกว่า 1 ตัวจนกลายเป็นเส้นสาย หรือเรียกอีกอย่างว่าลำดับของอักขระ (Array of Characters) โดยเราสามารถกำหนดค่าตัวแปรได้โดยใช้เครื่องหมาย

single quotation ('...') หรือ double quotation (" .... ") ครอบอักขระ, สายอักขระ หรือแม้แต่ตัวเลข และรวมไปถึงสัญลักษณ์พิเศษต่าง ๆ โดยในไพธอนนั้นจะเก็บข้อมูลแบบ 8-bit strings หรือ Unicode objects ขึ้นอยู่กับข้อมูลที่จัดเก็บอยู่ในขณะนั้น ดังตัวอย่างต่อไปนี้

```
>>> print "Hello, World!"
```

และได้ผลการทำงาน

```
Hello, World!
```

ซึ่งบางครั้งเราต้องการพิมพ์ค่าของตัวแปรนั้นไปพร้อมกับข้อความใน function print นั้นสามารถใช้สัญลักษณ์ , (Concatenation Symbol) ด้านหลังชุดข้อความนั้น ๆ ก่อน แล้วจึงพิมพ์ชื่อตัวแปรนั้น ๆ ต่อท้าย

```
>>> i = 5
>>> print "14 / 3 = ", 14 / 3
>>> print "14 % 3 = ", 14 % 3
>>> print "14.0 / 3.0 =", 14.0 / 3.0
>>> print "hello", "hello", i
```

ผลการทำงาน

```
14 / 3 = 4
14 % 3 = 2
14.0 / 3.0 = 4.6666666666667
hello hello 5
```

ซึ่งการใช้ Concatenation Symbol นั้นสามารถที่จะนำมาประยุกต์ใช้งาน อื่น ๆ ได้อีกนอกจาก print แต่ถ้าต้องการให้ข้อความนั้นต่อกันให้ใช้สัญลักษณ์ + (String Concatenation Symbol) เพื่อเชื่อมข้อความแต่ละชุดเข้าด้วยกัน

```
>>> print "Jack and Jill went up a hill" + "to fetch a pail of water;" +
"Jack fell down, and broke his crown," + "and Jill came tumbling after."
```

ผลการทำงาน

```
Jack and Jill went up a hill to fetch a pail of water;
Jack fell down, and broke his crown, and Jill came
tumbling after.
```

ซึ่งการใส่ข้อมูลภายใต้เครื่องหมาย double quotation นั้น สามารถใช้สัญลักษณ์พิเศษได้ดังนี้



<code>\newline</code>	ไม่สนใจ (Ignored)
<code>\\</code>	เครื่องหมาย Backslash ( \ )
<code>\'</code>	เครื่องหมาย Single quote ( ' )
<code>\"</code>	เครื่องหมาย Double quote ( " )
<code>\a</code>	ASCII Bell (BEL)
<code>\b</code>	ASCII Backspace (BS)
<code>\f</code>	ASCII Formfeed (FF)
<code>\n</code>	ASCII Linefeed (LF)
<code>\r</code>	ASCII Carriage Return (CR)
<code>\t</code>	ASCII Horizontal Tab (TAB)
<code>\v</code>	ASCII Vertical Tab (VT)

โดยเป็นสัญลักษณ์พิเศษที่เราเจอกันอยู่ในหลาย ๆ ภาษา

และในบางครั้งการต่อสายอักขระ นั้นอาจจะไม่สะดวกการใช้ String formatting operator "%" เข้ามาช่วยจะทำให้การนำตัวแปรต่าง ๆ มาใส่ในสายอักขระทำได้ง่ายมากขึ้น เช่น

```
>>> state = 'California'
>>> 'It never rains in sunny %s.' %state
```

จะได้ผลการทำงานคือ

```
'It never rains in sunny California.'
```

โดยที่ %s นั้นเราสามารถแทนด้วยตัวอักษร s เป็นตัวอักษรอื่น ๆ ได้ตามความเหมาะสมกับตัวแปรที่เราจะนำมาผสมลงในสายอักขระ ซึ่งมีดังนี้

d	Signed integer decimal
i	Signed integer decimal
o	Unsigned octal
u	Unsigned decimal
x	Unsigned hexadecimal (lowercase)
X	Unsigned hexadecimal (uppercase)
e	Floating point exponential format (lowercase)
E	Floating point exponential format (uppercase)
f	Floating point decimal format
F	Floating point decimal format
g	Same as "e" if exponent is greater than -4 or less than precision, "f" otherwise
G	Same as "E" if exponent is greater than -4 or less than precision, "F" otherwise
c	Single character (accepts integer or single character string)
r	String (converts any python object using repr())
s	String (converts any python object using str())

```

d = 4.5000
print 'It never rains in sunny %d.' %d
print 'It never rains in sunny %i.' %d
print 'It never rains in sunny %o.' %d
print 'It never rains in sunny %u.' %d
print 'It never rains in sunny %x.' %d
print 'It never rains in sunny %X.' %d
print 'It never rains in sunny %e.' %d
print 'It never rains in sunny %E.' %d
print 'It never rains in sunny %f.' %d
print 'It never rains in sunny %F.' %d
print 'It never rains in sunny %g.' %d
print 'It never rains in sunny %G.' %d
print 'It never rains in sunny %c.' %int(d)
print 'It never rains in sunny %r.' %d
print 'It never rains in sunny %s.' %d

```

#### ผลการทำงาน

```

It never rains in sunny 4.
It never rains in sunny 4.
It never rains in sunny 4.
It never rains in sunny 4.
It never rains in sunny 4.
It never rains in sunny 4.
It never rains in sunny 4.500000e+000.
It never rains in sunny 4.500000E+000.
It never rains in sunny 4.500000.
It never rains in sunny 4.500000.
It never rains in sunny 4.5.
It never rains in sunny 4.5.
It never rains in sunny L.
It never rains in sunny 4.5.
It never rains in sunny 4.5.

```

### 5.3.1 ฟังก์ชันที่น่าสนใจเกี่ยวข้องกับสายอักขระ

#### ฟังก์ชัน find

รูปแบบฟังก์ชัน `Object.find((sub[, start[, end]])` เป็นฟังก์ชันที่ใช้ในการค้นหาค่าในสายอักขระโดยจะคืนค่าใน หมายเลขสมาชิก (index key) ของสายอักขระนั้น ๆ ที่เจอเป็นตัวแรก แต่ถ้าไม่เจอจะส่งค่า -1 กลับมา

```
>>> s = "windows"  
>>> s.find('dow')  
3
```

### ฟังก์ชัน upper

รูปแบบฟังก์ชัน *Object.upper()* เป็นฟังก์ชันที่ใช้ในการทำให้ตัวอักษรเปลี่ยนเป็นตัวใหญ่ (ในภาษาอังกฤษ)

```
>>> s = "windows"  
>>> s.upper()  
WINDOWS
```

### ฟังก์ชัน lower

รูปแบบฟังก์ชัน *Object.lower()* เป็นฟังก์ชันที่ใช้ในการทำให้ตัวอักษรเปลี่ยนเป็นตัวเล็ก (ในภาษาอังกฤษ)

```
>>> s = "WINDOWS"  
>>> s.lower()  
windows
```

### ฟังก์ชัน replace

รูปแบบฟังก์ชัน *Object.replace(old, new)* เป็นฟังก์ชันที่ใช้ในการค้นหาคำที่กำหนดและแทนที่คำนั้นด้วยคำที่กำหนดให้

```
>>> s = 'The happy cat ran home.'  
>>> s.replace('cat', 'dog')  
'The happy dog ran home.'
```



# บทที่ 6

## การเปรียบเทียบ (Comparisons)

ในภาษาไพธอนนั้นมีการเปรียบเทียบในเชิงคณิตศาสตร์ต่าง ๆ เป็นพื้นฐานอยู่แล้ว แต่บางอย่างก็มีที่ไม่เหมือนกับภาษาอื่น ๆ คือมีการเพิ่มเติมการเปรียบเทียบในด้านของกลุ่มข้อมูล และข้อมูลที่เหมือนกัน โดยได้เพิ่ม การเปรียบเทียบแบบ "in" และ "is" เข้ามาเพื่อเปรียบเทียบกับกลุ่มข้อมูล ส่วนใหญ่การใช้ "is" มักใช้ในการเปรียบเทียบด้านตัวอักษรและสายอักขระมากกว่าเพราะสื่อความหมายมากกว่าใช้ "==" ในการเปรียบเทียบ

โดยเมื่อมีการเปรียบเทียบแล้วเราจะได้ออกมาคือค่าทางตรรกะ หรือ Boolean value ซึ่งในภาษาไพธอนนั้นคือ True แทนด้วยเป็นจริง และ False แทนด้วยเป็นเท็จ และเรายังสามารถใช้ตัวเลขแทนค่าดังกล่าวได้ด้วย 0 คือ False และ 1 คือ True นั้นเอง โดยค่าที่ได้นั้นมาจากการทำการทดสอบทางตรรกศาสตร์ในเชิงเปรียบเทียบค่าทั้งสองข้างของข้อมูล

Operator	ความหมาย
<	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย น้อยกว่า ค่าทางด้านขวาหรือไม่
<=	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย น้อยกว่าหรือเท่ากับ ค่าทางด้านขวา หรือไม่
>	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย มากกว่า ค่าทางด้านขวาหรือไม่
>=	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย มากกว่าหรือเท่ากับ ค่าทางด้านขวาหรือไม่
==	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย เท่ากับ ค่าทางด้านขวาหรือไม่
!=	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย ไม่เท่ากับ ค่าทางด้านขวาหรือไม่
<>	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย ไม่มีทางเท่ากับ ค่าทางด้านขวาหรือไม่
in	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมายอยู่ในกลุ่มข้อมูลในกลุ่มข้อมูลด้านขวาหรือไม่
is	เปรียบเทียบค่าทางด้านซ้ายของเครื่องหมาย เหมือนกับ ค่าทางด้านขวาหรือไม่

### ตัวอย่าง

```
>>> 1 < 2
True
>>> 2 <= 2
True
>>> 2 > 0
True
>>> 3 >= 3
True
>>> 0 == 0
True
>>> 0 != 1
False
>>> x = [1, 2, 3, 4]
>>> 1 in x
True
>>> 1 is 1
True
```



# บทที่ 7

## นิพจน์ทางตรรกศาสตร์ (Boolean Expressions)

การทำการเปรียบเทียบนั้น เมื่อได้ค่าจากการเปรียบเทียบซึ่งเป็นค่าทางตรรกะมาหนึ่งค่า โดยทั่วไปแล้วก็เพียงพอต่อการนำไปใช้ในการทำสอบทางตรรกศาสตร์ (Condition) อยู่แล้ว แต่บางครั้งแล้วเรามักนำค่าต่าง ๆ มาเชื่อมกันเพื่อให้ได้นิพจน์ที่มีความหมายเชื่อมโยงกันเพื่อให้ได้ความหมายที่ดีมากขึ้นเมื่อมีการทดสอบทางตรรกศาสตร์มากกว่า 1 ชุดการทดสอบ และเป็นการช่วยให้การเขียนโปรแกรมง่ายขึ้นด้วย โดยในไพธอนนั้นมีนิพจน์ทางตรรกศาสตร์ตามพื้นฐานภาษาทั่วไปคือ *AND*, *OR* และ *NOT* โดยเป็นการเปรียบเทียบจากผลของการเปรียบเทียบในเชิงคณิตศาสตร์ที่ได้มาก่อนแล้วมาเปรียบเทียบในเชิงตรรกศาสตร์อีกรอบหนึ่ง

### 7.1 AND (และ)

มีคีย์เวิร์ดคือ *and* จะประมวลผลประโยคหรือตัวแปรที่ตามหลังตัวมันเอง ถ้าประโยคหรือตัวแปรที่อยู่ก่อนหน้ามีค่าทางตรรกศาสตร์เป็น *false* เพราะว่าค่าที่เป็น *false* เมื่อนำมาประมวลผลกับ ค่าอื่น ๆ แล้วจะได้ *false* เสมอ

expression	ผล
true and true	True
true and false	False
false and true	False
false and false	False

### 7.2 OR (หรือ)

มีคีย์เวิร์ดคือ *or* จะประมวลผลประโยคหรือตัวแปรที่ตามมาเมื่อประโยคแรก หรือตัวแปรตัวแรกมีค่าเป็น *true* เพราะว่าประโยคหรือตัวแปรที่มีค่าเป็น *true* เมื่อนำมาประมวลผลกับค่าใด ๆ ก็ตามจะได้ค่า *true* เสมอ

expression	ผล
true or true	True
true or false	True
false or true	True
false or false	False

## 7.3 NOT (ไม่)

มีคีย์เวิร์ดคือ *not* โดยเมื่อไปอยู่หน้าตัวแปรหรือประโยคที่มีค่าทางตรรกศาสตร์ค่าใดค่าหนึ่ง ก็จะเปลี่ยนค่านั้นให้เป็นตรงกันข้ามทันที

expression	ผล
not true	False
not false	True

ตัวอย่าง

```
>>> i = 1
>>> j = 2
>>> k = 3
>>> l = 1
>>> i == k and j == i
False
>>> i == k or l == i
True
>>> not i == k and l == i
True
```



## บทที่ 8

# ช่วงของการทำงาน (Statement block) และ ช่วงชีวิตของตัวแปร (Life time หรือ Variable scope)

### 8.1 ช่วงของการทำงาน (Statement block)

ก่อนอื่นเราต้องทำความเข้าใจเกี่ยวกับช่วงของการทำงานของไพธอนเสียก่อน ยกตัวอย่าง ในภาษา C นั้นช่วงของการทำงานจะครอบด้วย เครื่องหมายปีกกาเปิดและปิด ..... แต่ใน Python ใช้ย่อหน้าแทน (indentation) เช่น

```
>>> x = 1
>>> if x == 1:
>>>     print "True"
>>> else:
>>>     print "False"
>>> print "xxx"
```

การจบ block ก็ดูได้จากย่อหน้า คำสั่ง if ก็อย่างที่เห็น if ตามด้วย เงื่อนไข จบด้วย : (colon) หลังจากนั้น สิ่งที่จะถูกทำเมื่อเงื่อนไขเป็นจริงก็จะอยู่ในย่อหน้าเอียงถัดมา ซึ่งโปรแกรมข้างบนก็ได้ผลออกมาเป็น

```
True
xxx
```

### 8.2 ช่วงชีวิตของตัวแปร (Life time หรือ Variable scope)

ในการกำหนดช่วงชีวิตของตัวแปรว่าตัวแปรตัวไหนจะมีช่วงการทำงานในส่วนใดได้บ้างนั้นสามารถทำได้ โดยจากตัวอย่างด้านล่างนั้น เราได้สร้างตัวแปรชนิดจำนวนเต็มชื่อว่า x ให้ค่าคือ 5 และสร้างฟังก์ชันชื่อว่า hello

โดยในตัวอย่างที่ 1 นั้นเรากำหนดตัวแปรภายในฟังก์ชันให้มีชื่อเป็น x เหมือนกัน แล้วใส่ค่าให้เป็น 6 แล้วต่อมาให้ฟังก์ชัน hello ทำงาน และ print ค่า x ออกมา ผลคือได้ 6 และ 5 ตามลำดับ ในตัวอย่างที่ 2 นั้นเราได้กำหนดตัวแปร และฟังก์ชันต่าง ๆ เหมือนกับตัวอย่างที่ 1 เพียงแต่เราใช้ global มากำหนดให้กับ x ว่าเราจะใช้ x จาก global scope แทน แล้วทำการใส่ค่าให้กับ x เป็น 6 ผลที่ได้คือ 6 และ 6 ตามลำดับ

## ตัวอย่างที่ 1

```
>>> x = 5
>>> def hello():
>>>     x = 6
>>>     print x
>>> hello()
>>> print x
```

คำตอบคือ

6  
5

## ตัวอย่างที่ 2

```
>>> x = 5
>>> def hello():
>>>     global x
>>>     x = 6
>>>     print x
>>> hello()
>>> print x
```

คำตอบคือ

6  
6

จากตัวอย่างทั้งสองจะเห็นว่าเราสามารถในตัวแปรแบบภายในฟังก์ชันและจากภายนอกฟังก์ชันได้ด้วยวิธีดังต่อไปนี้

1. ถ้าต้องการใช้ตัวแปรภายนอกฟังก์ชันที่มีอยู่แล้วในคีย์เวิร์ด *global* แล้วตามด้วยชื่อตัวแปรที่มีอยู่แล้วจากภายนอกฟังก์ชันนั้น แล้วจึงนำมาใช้งาน
2. ถ้าต้องการประกาศตัวแปรใหม่ภายในฟังก์ชัน สามารถประกาศตัวแปรได้โดยทั่วไปได้ทันที
3. ถ้าต้องการใช้ทั้งตัวแปรภายนอกและภายในพร้อม ๆ กันให้ตั้งชื่อตัวแปรที่ตั้งใหม่ ซึ่งใช้ภายในฟังก์ชันนั้นให้ชื่อแตกต่างกันเพื่อป้องกันการสับสน

## บทที่ 9

# การควบคุมทิศทางของโปรแกรม (Control flow, Flow of Control หรือ Alternatively)

การควบคุมทิศทางของโปรแกรม เป็นการเลียนแบบ การทำงานของมนุษย์ เพราะในเหตุการณ์ต่าง ๆ เรา มีการตัดสินใจในแบบต่าง ๆ กันออกไป เช่นเมื่อเดินทางไปเจอ 3 แยก เราต้องตัดสินใจ เลี้ยวซ้ายหรือ เลี้ยวขวา โดยมีแผนที่ และป้ายบอกเส้นทางเป็นตัวกำหนด ให้เราเลี้ยวซ้ายหรือเลี้ยวขวา เป็นต้น ในการเขียน โปรแกรมก็คือ การจำลองตัวเราลงไป เพื่อจัดการกับสถานการณ์ต่างๆ ซึ่งเราต้องมีข้อมูลในการประกอบการ พิจารณา ถ้าข้อมูลบอกเราแบบหนึ่ง เราก็ต้องทำแบบหนึ่ง แต่ถ้าข้อมูลบอกเราอีกอย่าง เราก็ต้องทำอีกอย่าง ที่แตกต่างกันออกไป ซึ่งการที่ต้องจัดการกับสถานการณ์ต่างๆ นี้ เราเรียกมันว่าการควบคุมการทำงานของ โปรแกรมนั่นเอง การควบคุมการทำงาน มี 3 รูปแบบ ได้แก่

1. การตัดสินใจ (Decisions, Choice หรือ Selection)
2. การวนซ้ำ (Loop หรือ Iteration)
3. การจัดการความผิดปกติของโปรแกรม (Error Checking)

ในการควบคุมทิศทาง ไม่ว่าจะเป็นการตัดสินใจ หรือการทำงานแบบวนซ้ำ เราจะต้องอาศัยการ พิจารณาข้อมูล ที่มีอยู่ ประกอบการควบคุม ส่วนการจัดการความผิดปกติของโปรแกรมนั้นเป็นการดักจับสิ่งที่จะเกิดขึ้น ได้จากการเขียนโปรแกรมที่ไม่ครอบคลุมการทำงาน ซึ่งอาจจะเกิดขึ้นได้ แต่ป้องกันโปรแกรมปิดตัวเองโดย ฉับพลัน เราจึงใช้การจัดการความผิดปกติของโปรแกรมมาช่วยในการดักจับสิ่งเหล่านี้

## 9.1 การตัดสินใจ (Decisions, Choice หรือ Selection)

### 9.1.1 if Statements

คำสั่ง if ใช้ตัดสินใจว่าจะทำหรือไม่ทำคำสั่งชุดหนึ่งที่อยู่ภายในช่วงของการทำงานของ if (if Statements scope) ถ้าเงื่อนไขที่นำมาทดสอบทางตรรกศาสตร์เป็นจริง (True) ก็จะทำคำสั่งชุดนั้นถ้าเงื่อนไขที่นำมาทดสอบ ทางตรรกศาสตร์เป็นเท็จ (False) ก็จะไม่ทำคำสั่งชุดนั้น โดยมีรูปแบบคำสั่งในภาษาไพธอนดังนี้

```

if Condition:
    Statements
    .....
    .....
else:
    Statements
    .....
    .....

```

### ตัวอย่าง

```

>>> x = 1
>>> if x is 1:
>>>     print 'Yes'
>>> else:
>>>     print 'No'

```

โดยจากตัวอย่างด้านบนนั้นจะได้ผลออกมาเป็น "Yes" เนื่องจากว่า x มีค่าเป็น 1 และเปรียบเทียบกับ 1 ซึ่งมีค่าเดียวกันผลที่ออกมาเป็น True ในการเปรียบเทียบ แล้วจึงพิมพ์ค่า "Yes" ออกมานั่นเอง

ซึ่งในบางครั้งการตัดสินใจในการทำชุดคำสั่งอาจมีมากกว่า 2 ทางจากข้างต้น เราสามารถในคำสั่งอีกหนึ่งตัวที่ชื่อว่า *elif* ได้ (หรือ มาจาก *else if* ในภาษาอื่น เช่น java หรือ c/c++) โดยมีรูปแบบดังต่อไปนี้

```

if Condition:
    Statements
    .....
    .....
elif Condition:
    Statements
    .....
    .....
else:
    Statements
    .....
    .....

```

### ตัวอย่าง

```

>>> x = 3
>>> if x < 1:
>>>     print 'x < 1'
>>> elif x > 1:
>>>     print 'x > 1'
>>> else:
>>>     print 'x = 1'

```

จากตัวอย่างด้านบน นั้นเราจะได้คำตอบคือ "x > 1"

### 9.1.2 switch Statements

ในไพธอนนั้นไม่สนับสนุนการตัดสินใจแบบ switch

## 9.2 การวนทำซ้ำ (Loop)

### 9.2.1 while Statements

คำสั่ง while เป็นคำสั่งที่ใช้ในการวนทำซ้ำในช่วงของการทำงานของ while (while Statements scope) จนกว่าการทดสอบทางตรรกศาสตร์จะเป็นเท็จ มีรูปแบบดังนี้

```
while Condition:
    Statements
    .....
    .....
```

ตัวอย่าง

```
>>> x = 1
>>> while x < 5:
>>>     print 'Yes\n'
>>>     x+=1
```

ผลจากการทำ

```
Yes
Yes
Yes
Yes
```

จากตัวอย่างคือกำหนดให้ x มีค่าเป็น 1 แล้วเปรียบเทียบทางตรรกศาสตร์ก่อน โดยให้  $x < 5$  ในที่นี้ x มีค่าเป็น 1 ผลคำตอบคือ True จึงทำคำสั่งภายในช่วงการทำงานของ while โดยการพิมพ์ Yes แล้วทำการเพิ่มค่า x โดยการบวกไปที่ละ 1 ค่า แล้วกลับมาเปรียบเทียบทางตรรกศาสตร์ใหม่ ทำอย่างนี้ไปเรื่อย ๆ จนกว่าการเปรียบเทียบทางตรรกศาสตร์จะมีผลเป็น False

### 9.2.2 for Statements

คำสั่ง for เป็นคำสั่งที่ใช้ในการวนทำซ้ำในช่วงของการทำงานของ for (for Statements scope) จนกว่าการทดสอบทางตรรกศาสตร์จะเป็นเท็จ โดยมีการกำหนดช่วงการทำงานต่ำสุด และมากที่สุดของจำนวนครั้งที่ทำคำสั่งภายในด้วย โดยใช้ for ร่วมกับคำสั่งในการเปรียบเทียบทางตรรกศาสตร์คือ in และ function การทำงานแบบช่วงของกลุ่มของข้อมูลคือ range (ดูวิธีการใช้ได้จากบทก่อน ๆ )

```
for var in range(m, n [, step = 1]):
    Statements
    .....
    .....
```

ตัวอย่าง

```
>>> for i in range(0, 10):
>>>     if i % 2 == 0:
>>>         print i, 'is an even number'
>>>     else:
>>>         print i, 'is an odd number'
```

จากตัวอย่างด้านบนนั้นทำการทำซ้ำตั้งแต่ 0 ไปจนถึง 9 แล้วทำการตรวจสอบหาเลขคู่และเลขคี่ด้วย

```
0 is an even number
1 is an odd number
2 is an even number
3 is an odd number
4 is an even number
5 is an odd number
6 is an even number
7 is an odd number
8 is an even number
9 is an odd number
```

ซึ่งในไพธอนนั้นไม่สนับสนุน *foreach* แต่เราสามารถใช้ความสามารถของ *for* มาใช้งานจนได้คำสั่ง *foreach* ได้อยู่แล้ว

```
>>> x = {'first': 'one', 'second': 'two'}
>>> for j in x:
>>>     print x[j]
two
one
```

### 9.2.3 pass, break, continue และ else Clauses Statements

**pass**

เป็นคำสั่งที่ใช้ในการทดแทน Statements อื่น ๆ โดยไม่มีการทำงานตัวอย่าง

```
>>> while True:
>>>     pass
>>>
```

**break**

คำสั่งนี้ใช้ในการหยุดการทำงานของลูปการวนซ้ำ หรือการตัดลูป

**continue**

คำสั่งนี้ใช้ในการเริ่มการทำงานหลังจากหยุดการทำงานของลูปการวนซ้ำ หรือการตัดลูป

**else Clauses**

คำสั่งนี้ใช้ในการทำหลังจาก break ใน การวนซ้ำ (Loop, Iteration)

ตัวอย่าง

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        print n, 'is a prime number'
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

**9.2.4 do-while Statements**

ในไพธอนนั้นไม่สนับสนุนการตัดลูปแบบ do-while (จะสนับสนุนบน Python 2.5 และมีบน Python 2.5 Beta เมื่อวันที่ 24 กุมภาพันธ์ 2549)

**9.3 การจัดการความผิดปกติของโปรแกรม (Error Checking)****9.3.1 assert Statements**

`assert` เป็นคำสั่งที่วางไว้สำหรับตรวจสอบความผิดพลาดภายในคำสั่งที่เราเขียนในโปรแกรมของเรา โดยเป็นเหมือนส่วนเติมเต็มในการแจ้งความผิดพลาดของตัวไพธอนเพื่อบอกรายละเอียดที่มากขึ้นในการเขียนโปรแกรมของเรา ดังรูปแบบคือ

```
assert Condition, 'Text Error or Text Mixed'
```

ตัวอย่าง

```
def test(arg1, arg2):
    arg1 = float(arg1)
    arg2 = float(arg2)
    assert arg2 != 0, 'Bad dividend, arg1: %f arg2: %f' % (arg1, arg2)
    ratio = arg1 / arg2
    print 'ratio:', ratio
test(0,2)
test(2,0)
```

เมื่อเราสั่งให้ทำงานจะได้ดังนี้

```
ratio: 0.0
Traceback (most recent call last):
  File "C:\tmp.py", line 51, in ?
test(2,0)
  File "C:\tmp.py", line 46, in test
assert arg2 != 0, 'Bad dividend, arg1: %f arg2: %f' % (arg1, arg2)
AssertionError: Bad dividend, arg1: 2.000000 arg2: 0.000000
```

จะเห็นได้ว่าโปรแกรมจะมีตัวช่วยในการสร้างมุมมองของตัวแปรต่าง ๆ ได้ว่าค่าที่ใส่ไปนั้นถูกต้องหรือไม่

### 9.3.2 try-except และ raise Statements (Exception handling)

ในภาษาไพธอนนั้น try-except เป็นการบอกถึงสิ่งผิดปกติที่เกิดขึ้นในโปรแกรม ซึ่งอาจเป็นข้อผิดพลาดหรือเหตุการณ์ที่ไม่พึงประสงค์และต้องการความดูแลเป็นพิเศษ โดยทั่วไปในการเขียนโปรแกรมที่ดีนั้น เราจะต้องตรวจสอบถึงเหตุการณ์ที่อาจทำให้โปรแกรมของเราล้มเหลวในการทำงาน เช่น ข้อมูลถูกหารด้วยศูนย์, การค้นหาข้อมูลใน list ด้วยการใช้อินดิกซ์ที่ไม่มีอยู่จริง หรือ อ้างถึงหน่วยความจำที่เป็น null เป็นต้น ถึงแม้ว่าเรามีวิธีการตรวจสอบ error ต่าง ๆ เหล่านี้ด้วยการใช้การตั้งสติใจด้วย if Statements หรือ การตรวจสอบอื่น ๆ ที่ทำให้โปรแกรมของเราทำงานได้ราบรื่น แต่จะทำให้ code ของเรายุ่งเหยิงเพราะถ้ามีการตรวจสอบข้อผิดพลาดมากเกินไปชุดคำสั่งของเราก็จะดูซับซ้อนมากยิ่งขึ้น แต่ไม่ได้หมายความว่าเราจะไม่ตรวจสอบ และดักจับข้อผิดพลาดเหล่านี้ การนำเอา try-except เข้ามาเป็นตัวช่วยในการตรวจสอบและดักจับ ทำให้เกิดการแยกส่วนของ code ที่ทำงานได้ราบรื่น ออกจากส่วนของชุดคำสั่งที่จัดการเกี่ยวกับความผิดพลาดทำให้เราสามารถที่จะค้นหาส่วนของคำสั่งทั้งสองได้ง่ายขึ้น ถ้ามีการเปลี่ยนแปลง code ในอนาคต ข้อดีอีกอันหนึ่งของ try-except ก็คือ ทำให้การตรวจสอบและดักจับเป็นไปอย่างเฉพาะเจาะจง ตรงกับข้อผิดพลาดที่เกิดขึ้น ทำให้การแก้ไขเป็นไปอย่างถูกต้อง และเนื่องจากว่าข้อผิดพลาดต่าง ๆ มีหลากหลายรูปแบบ เราต้องเขียนคำสั่งต่าง ๆ ขึ้นมารองรับไม่เช่นนั้นแล้ว โปรแกรมของเราก็จะไม่ผ่านการ compile เราไม่จำเป็นที่จะต้องให้ try-except ในการตรวจสอบและดักจับ error ในโปรแกรมเสมอไป เพราะการใช้ exception จะเสียเวลาในการประมวลผลมาก ทำให้โปรแกรมทำงานช้าลง ดังนั้นเราจะต้องตัดสินใจให้ดีกว่าควรจะใช้ try-except ในกรณีไหน อย่างไร ตัวอย่างของโปรแกรมที่ไม่ต้องใช้ try-except ก็น่าจะเป็นการใส่ข้อมูลนำเข้าแบบผิด ๆ ของผู้ใช้ ซึ่งถือเป็นเรื่องปกติ ถ้าเรามัวแต่เสียเวลาในการดักจับด้วยการใช้ try-except แทนการตรวจสอบและดักจับทั่วไปโปรแกรมของเราก็จะเสียเวลาในการประมวลผลส่วนอื่น ๆ ไป



### รูปแบบคำสั่ง

```
try:
    Standard operation
except:
    Error operation
```

จากรูปแบบคำสั่ง ชุดคำสั่งเดิมหรือคำสั่งที่ต้องการทำงานทั่ว ๆ ไป จะอยู่ภายใต้ช่วงการทำงานของ *try* ถ้าการทำงานภายในช่วงการทำงานของ *try* มีข้อผิดพลาดการทำงานจะกระโดด หรือถูกโยนการทำงานไปสู่ช่วงการทำงานของ *except* แทน ดังตัวอย่างด้านล่างนี้

```
>>> try:
...     x = y
... except:
...     print 'y not defined'
...
y not defined
```

ถ้าเราไม่ได้ใช้ *try-except* จะได้ผลแบบนี้

```
>>> x = y
Traceback (most recent call last):
  File "<input>", line 1, in ?
NameError: name 'y' is not defined
```

จากตัวอย่างที่ได้ดูไปเป็นรูปแบบ *try-except* ที่ง่ายที่สุด แต่ในการดักจับข้อผิดพลาดจริง ๆ แล้วเราไม่สามารถที่จะทำแบบนี้ได้ทุกกรณี เพราะข้อผิดพลาดในชุดคำสั่งที่เราเขียนไปนั้นมีย่อยของความผิดพลาดที่แตกต่างกัน เช่นความผิดพลาดของการอ่านและเขียนไฟล์, ข้อผิดพลาดจากการทำเกินขอบเขตของลิสต์, ฯลฯ โดยเราได้จากตัวอย่างต่อไปนี้

ตัวอย่างที่ไม่ได้ใช้ *try-except*

```
>>> fridge_contents = {"egg":8, "mushroom":20, "pepper":3, "cheese":2,
"tomato":4, "milk":13}
>>> if fridge_contents["orange juice"] > 3:
...     print "Sure, let's have some juice"
...
Traceback (most recent call last):
File "<stdin>", line 1, in ?
KeyError: 'orange juice'
```

จากด้านบนไม่ได้มีการแก้ไขปัญหาทาง *try-except* ทำให้แจ้งข้อผิดพลาดแบบควบคุมไม่ได้ เราจึงเขียนใหม่โดยใช้ *try-except* มาแก้ปัญหานี้

```
>>> fridge_contents = {"egg":8, "mushroom":20, "pepper":3, "cheese":2,
"tomato":4, "milk":13}
>>> try:
```

```

...     if fridge_contents["orange juice"] > 3:
...         print "Sure, let's have some juice"
...     except KeyError:
...         print "Aww, there's no juice. Lets go shopping"
...
Aww, there's no juice. Lets go shopping

```

เมื่อเราใช้ *try-except* เราสามารถควบคุมควบคุมว่าผลของความผิดพลาดที่ออกมาจะเป็นอย่างไรด้วย โดยควบคุมชนิดของความผิดพลาดว่าจะให้ออกมาเป็นแบบใด ในที่นี้ในการ *except* ความผิดพลาดแบบ *KeyError* แล้วให้พิมพ์ข้อความผิดพลาดออกมา

```

>>> fridge_contents = {"egg":8, "mushroom":20, "pepper":3, "cheese":2,
"tomato":4, "milk":13}
>>> try:
...     if fridge_contents["orange juice"] > 3:
...         print "Sure, let's have some juice"
...     except KeyError, error:
...         print "Woah! There is no %s" % error
...
Woah! There is no 'orange juice'

```

*raise* เป็น Statements เสริมของ *try-except* เพื่อช่วยในการโยนความผิดพลาดไปใช้ใน *try-except* ต่ออื่น ๆ ที่เกี่ยวข้อง

```

>>> class E(RuntimeError):
...     def __init__(self, msg):
...         self.msg = msg
...     def getMsg(self):
...         return self.msg
...
>>>
>>> try:
...     raise E('my test error')
... except E, obj:
...     print 'Msg:', obj.getMsg()
...
Msg: my test error

```

# บทที่ 10

## การสร้างฟังก์ชัน (Defined Function)

ฟังก์ชันคือแหล่งรวมชุดคำสั่งหลาย ๆ โดยคำสั่งที่เราเรียบเรียงขึ้นเอง เพื่อนำไปใช้ในการเขียนโปรแกรมซ้ำ ๆ กันโดยไม่ต้องเขียนชุดคำสั่งนั้น ๆ ใหม่อีกครั้ง (Reusability Code) โดยในไพธอนนั้นก็มีฟังก์ชันอยู่ 2 แบบคือ สามารถคืนค่ากลับมาได้ (real function, return value) และแบบไม่คืนค่า (void, sub , subprogram หรือ subroutine ) การสร้างฟังก์ชันจะใช้คีย์เวิร์ดชื่อ def แล้วตามด้วยชื่อของฟังก์ชันนั้น ๆ โดยลักษณะของช่วงการทำงานเหมือนกับคำสั่งควบคุมทิศทางโปรแกรมต่าง ๆ เช่นเดียวกัน

รูปแบบการเขียนคำสั่ง

```
def function_name( [Argument] ) :  
    Statement  
    .....  
    [return]
```

- **function\_name** ชื่อฟังก์ชัน
- **Statements** ชุดคำสั่ง
- **var** ตัวแปร
- **Argument** รับค่าของฟังก์ชัน
- **return** การคืนค่ากลับ โดยที่การคืนค่ากลับนั้นสามารถคืนค่ากลับได้มากกว่า 1 ค่า หรือมากกว่า 1 ตัวแปร ซึ่งสามารถคืนค่าได้ทุกชนิดข้อมูลด้วย

ตัวอย่าง

```
# ส่วนนี้ใช้ define ฟังก์ชัน  
def foo() :  
    print "Foobar"  
# ส่วนนี้เป็นส่วนที่เรียกใช้ ต้องมีเครื่องหมาย () หลังวงเล็บเสมอ  
foo()
```

ผลการทำงาน

Foobar

## 10.1 การรับค่าของฟังก์ชัน, คืนค่ากลับ และค่ามาตรฐานของการรับค่า

1. การรับค่าของฟังก์ชัน (Function Argument)
2. การคืนค่ากลับ (return)
3. ค่ามาตรฐานของการรับค่า (Function Default Argument)

ตัวอย่าง

```
# ส่วนนี้ใช้ define ฟังก์ชัน
def foo(text):
    print text
# ส่วนนี้เป็นส่วนที่เรียกใช้ ต้องมีเครื่องหมาย () หลังวงเล็บเสมอ
foo("Foobar")
```

ผลการทำงาน

Foobar

ตัวอย่าง คีย์เวิร์ด *return* เป็นคำสั่งให้ฟังก์ชันนั้นคืนค่าและ ออกจาก function

```
def add(x,y):
    return x + y
print add(10,20)
print add(add(10,20),30)
```

ผลการทำงาน

30

60

ตัวอย่าง การคืนค่าแบบหลายค่า หรือหลายตัวแปร

```
a, b, c = 0, 0, 0
def getabc():
    a = "Hello"
    b = "World"
    c = "!"
    return a,b,c
def gettuple():
    a,b,c = 1,2,3
    return (a,b,c)
def getlist():
    a,b,c = (3,4), (4,5), (5,6)
    return [a,b,c]
a,b,c = getabc()
```

```
print a,b,c
d,e,f = gettuple()
print d,e,f
g,h,i = getlist()
print g,h,i
```

**ผลการทำงาน**

```
Hello World !
1 2 3
(3, 4) (4, 5) (5, 6)
```

ตัวอย่าง กำหนดค่ามาตรฐานของตัวแปร x และ y ให้มีค่าเป็น 0 เพื่อป้องกันการไม่ใส่ให้กับตัวรับค่าในฟังก์ชัน

```
def add(x = 0,y = 0):
    return x + y
print add(10,20)
print add()
```

**ผลการทำงาน**

```
30
0
```

**ตัวอย่าง**

```
def multiprint( n=5, txt="Hello" ):
    i = 0
    while i < n:
        print txt
```

```
multiprint()
```

**ผลการทำงาน**

```
Hello
Hello
Hello
Hello
```

**ตัวอย่าง**

```
def factorial(n = 1):
    if n <= 1:
        return 1
    return n*factorial(n-1)
print "2! = ",factorial(2)
```

```
print "3! = ", factorial(3)
print "4! = ", factorial(4)
print "5! = ", factorial(5)
```

ผลการทำงาน

```
2! = 2
3! = 6
4! = 24
5! = 120
```

## 10.2 ตัวแปรแบบ Global (ทั่วไป) และ Local (เฉพาะส่วน)

ในการประกาศตัวแปรตัวหนึ่งขึ้นมาใช้ จะเปลืองหน่วยความจำไปส่วนหนึ่งสำหรับเก็บค่าตัวแปร ดังนั้นถ้าเราประกาศตัวแปรมา 1 ตัว แล้วนำไปใช้เพียงในฟังก์ชันเดียว จะเป็นการสิ้นเปลืองโดยใช้เหตุ ดังนั้นควรประกาศตัวแปรแบบทั่วไปบ้างตามความเหมาะสม

โดยตัวแปรเฉพาะส่วนที่อยู่ภายในฟังก์ชันนั้น เมื่อฟังก์ชันจบการทำงานตัวแปรพวกนี้จะถูกลบออกไปจากหน่วยความจำทันที ซึ่งประโยชน์อีกอย่างหนึ่งของตัวแปรเฉพาะ (Local Variable) ก็คือ ส่วนอื่น ๆ ของ โปรแกรมจะไม่รู้จักตัวแปรเฉพาะส่วนที่อยู่ในฟังก์ชันเลย ดังนั้นเราก็สามารถใช้ตัวแปรชื่อเดียวกันได้พร้อมกันในคนละส่วนของโปรแกรม โดยไม่เกิดข้อผิดพลาด

โดยในการทำงานหลัก ๆ ของโปรแกรมเราก็ต้องใช้ตัวแปรทั่วไป (Global Variable) โดยที่ตัวแปรทั่วไปจะเป็นที่รู้จักไปทั้งโปรแกรมดังนั้นฟังก์ชันต่าง ๆ ก็สามารถใช้ได้ด้วย โดยการเรียกใช้ตัวแปรทั่วไปในนั้นต้องใช้ดีเลย์เวิร์ดที่ชื่อ *global* ตามด้วยชื่อตัวแปรแบบทั่วไป

ตัวอย่าง การใช้ตัวแปรแบบเฉพาะที่

```
x = 5
def hello():
    x = 6
    print x
hello()
print x
```

ผลการทำงาน

```
6
5
```

ตัวอย่าง การใช้ตัวแปรแบบทั่วไป

```
x = 5
def hello():
    global x
    x = 6
    print x
hello()
print x
```

ผลการทำงาน

```
6
6
```

# บทที่ 11

## การใส่ข้อมูลผ่านคีย์บอร์ด (Input Data from Keyboard)

เราสามารถให้ผู้ใช้สามารถที่จะใส่ค่าที่เราต้องการได้ผ่านทางคีย์บอร์ดโดยทำงานผ่านฟังก์ชันที่ชื่อว่า *raw\_input* รูปแบบการเขียนคำสั่ง

```
var = raw_input([prompt])
```

- prompt ข้อความที่เป็นคำถาม
- var ตัวแปรที่มารับข้อมูล ซึ่งค่าที่ส่งออกมาจาก *raw\_input* ฟังก์ชัน โดยมีชนิดข้อมูลเป็น String

ตัวอย่าง

```
print "Halt!"  
s = raw_input("Who Goes there? ")  
print "You may pass,", s
```

เมื่อสั่งทำงานจะแสดงข้อความดังนี้

```
Halt!  
Who Goes there?
```

ทำการกรอกข้อมูลลงไป ในตัวอย่างนี้กรอกคำว่า Josh ลงไป แล้วจะทำการแสดงออกมา

```
Halt!  
Who Goes there? Josh  
You may pass, Josh
```

## ตัวอย่าง

```
menu_item = 0
list = []
while menu_item != 9:
    print "-----"
    print "1. Print the list"
    print "2. Add a name to the list"
    print "3. Remove a name from the list"
    print "4. Change an item in the list"
    print "9. Quit"
    menu_item = input("Pick an item from the menu: ")
    if menu_item == 1:
        current = 0
        if len(list) > 0:
            while current < len(list):
                print current, ". ", list[current]
                current = current + 1
        else:
            print "List is empty"
    elif menu_item == 2:
        list.append(raw_input("Type in a name to add: "))
    elif menu_item == 3:
        del_name = raw_input("What name would you like to remove: ")
        if del_name in list:
            item_number = list.index(del_name)
            del list[item_number]
            #The code above only removes the first occurrence of
            # the name. The code below from Gerald removes all.
            #while del_name in list:
            #    item_number = list.index(del_name)
            #    del list[item_number]
        else:
            print del_name, " was not found"
    elif menu_item == 4:
        old_name = raw_input("What name would you like to change: ")
        if old_name in list:
            item_number = list.index(old_name)
            list[item_number] = raw_input("What is the new name: ")
        else:
            print old_name, " was not found"
print "Goodbye"
```



การทำงาน เมื่อทำงานจะแสดงข้อความด้านล่างนี้

-----

1. Print the list
2. Add a name to the list
3. Remove a name from the list
4. Change an item in the list
9. Quit

ให้พิมพ์ 2 ลงไปแล้วกรอกคำว่า Jack และไล่การทำงานต่าง ๆ ดังตัวอย่างด้านล่าง ไปเรื่อย ๆ ซึ่งนี่คือการทำงานโดยการรับค่าผ่านทางคีย์บอร์ดนั่นเอง

Pick an item from the menu: 2  
Type in a name to add: Jack

Pick an item from the menu: 2  
Type in a name to add: Jill

Pick an item from the menu: 1  
0 . Jack  
1 . Jill

Pick an item from the menu: 3  
What name would you like to remove: Jack

Pick an item from the menu: 4  
What name would you like to change: Jill  
What is the new name: Jill Peters

Pick an item from the menu: 1  
0 . Jill Peters

Pick an item from the menu: 9  
Goodbye



## ภาคผนวก ก

# เรื่องที่ห้ามลืมใน Python

1. **Don't forget the colons** อย่าลืมโคลอน (Colon, :) เมื่อจบคำสั่งควบคุมทิศทางของโปรแกรมต่าง ๆ เช่น if, while หรือ for เป็นต้น
2. **Start in column 1** ไพธอนใช้ระบบแท็บแทนปีกกาเพื่อควบคุมช่วงของการทำงาน และต้องเริ่มจากคอลัมน์ที่ 1 ทุกครั้งด้วย
3. **Blank lines matter at the interactive prompt** อย่าเผลอมีบรรทัดว่างหรือขึ้นบรรทัดใหม่ใน shell prompt ถ้าไม่ชัวร์ว่าเขียนคำสั่งจบในบรรทัดนั้น ๆ
4. **Indent consistently** อย่าใช้ปุ่มแท็บปนกับการเคาะเว้นวรรค โดยควรเลือกว่าจะใช้การแท็บหรือเคาะวรรค ถ้าใช้เคาะวรรคก็ควรใช้ให้ตลอดรอดฝั่ง โดย 1 แท็บ ให้มีขนาดเท่ากับเคาะวรรค 4 ครั้ง (4 whitespaces/tab)
5. **Don't code C in Python** อย่าเขียนโค้ดแบบ C ในไพธอน เช่น if (X==1): print X โดยในความเป็นจริงแล้วไม่ต้องมี () ก็ได้ เป็นต้น
6. **Don't always expect a result** บาง Method เช่น Append หรือ Sort อย่าไปคิดว่ามันจะ return obj บางครั้งมัน return None หรือ Null ออกมา ซึ่งเราไม่จำเป็นต้องเขียน list=list.append(X) แต่ให้เขียน list.append(X) ลงไปได้เลย
7. **Use calls and imports properly** หลังเรียก Method ให้มี () ด้วยเช่น function() อย่าใช้ function เฉยๆ และตอน Import ไม่ต้องใส่นามสกุล file อย่าง import mod.py ใช้ import mod เฉยๆ ก็พอ



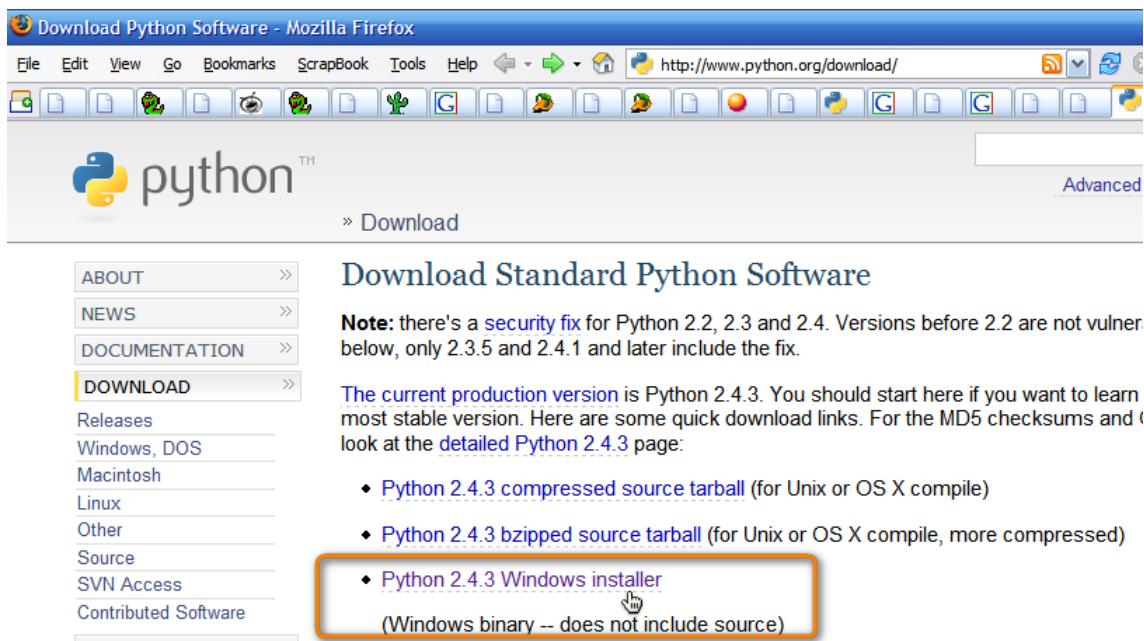
## ภาคผนวก ข

# การติดตั้ง Python, wxPython และ Stani's Python Editor

### ข.1 การติดตั้ง Python

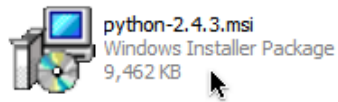
\* ในเอกสารนี้อ้างอิง Python Version 2.4.3

1. ดาวน์โหลด Python 2.4 สำหรับ Windows ได้จาก <http://www.python.org/download/> และเลือก version 2.4.x จากรายการดาวน์โหลด โดยเลือกรูปแบบ "Python 2.4.x Windows installer"



รูปที่ ข.1: เลือกดาวน์โหลด Python 2.4 สำหรับ Windows

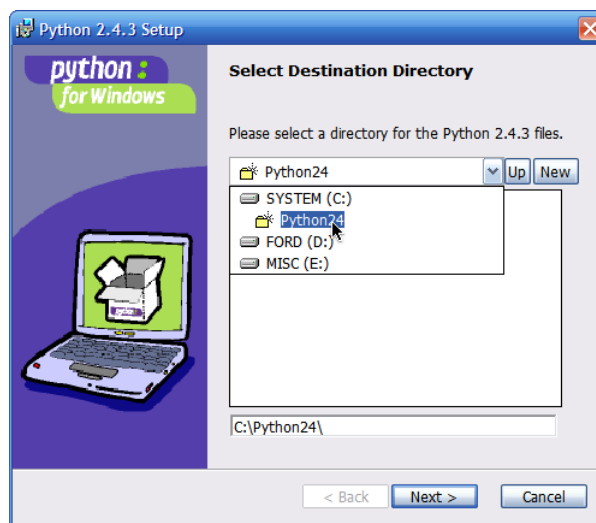
2. ดับเบิลคลิกที่ Python-2.xxx.yyy.exe ที่เป็นไฟล์ที่เราดาวน์โหลด แล้วทำตามขั้นตอนการติดตั้งดังต่อไปนี้



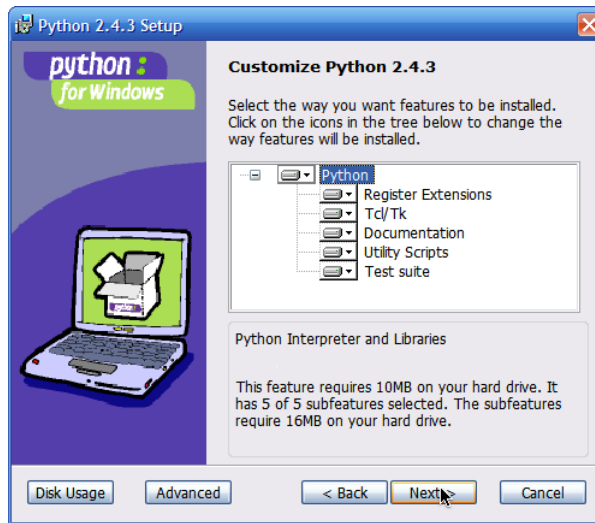
รูปที่ ข.2: ขั้นตอนที่ 1 : ดับเบิลคลิกที่ไฟล์ติดตั้ง



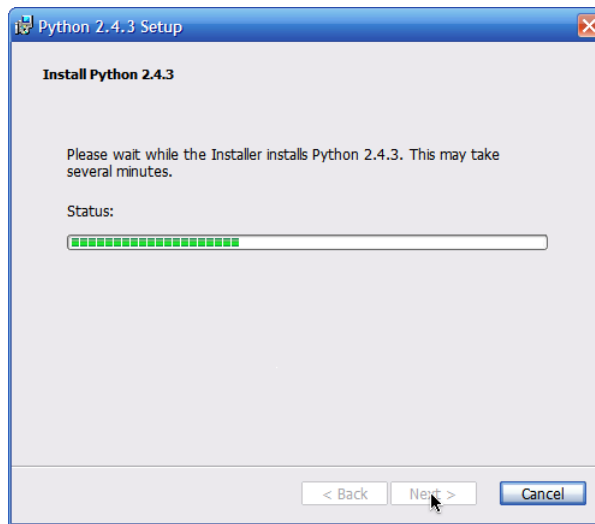
รูปที่ ข.3: ขั้นตอนที่ 2 : เลือก "Install for all users"



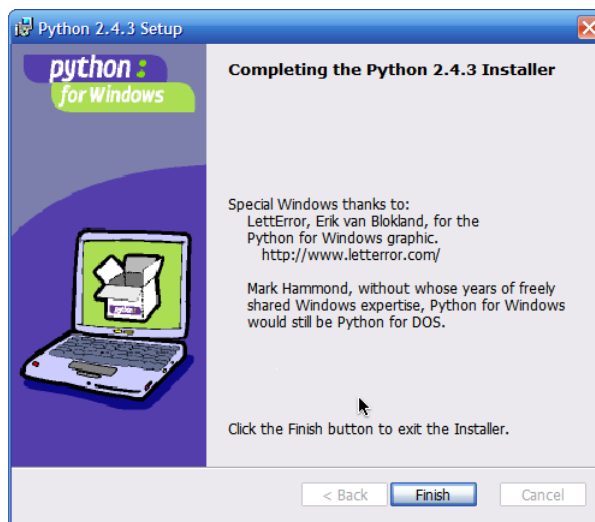
รูปที่ ข.4: ขั้นตอนที่ 3 : ให้เลือกที่ติดตั้งที่ C:\Python24\



รูปที่ ข.5: ขั้นตอนที่ 4 : เลือกติดตั้งทุกตัวเลือก



รูปที่ ข.6: ขั้นตอนที่ 5 : ดำเนินการติดตั้ง

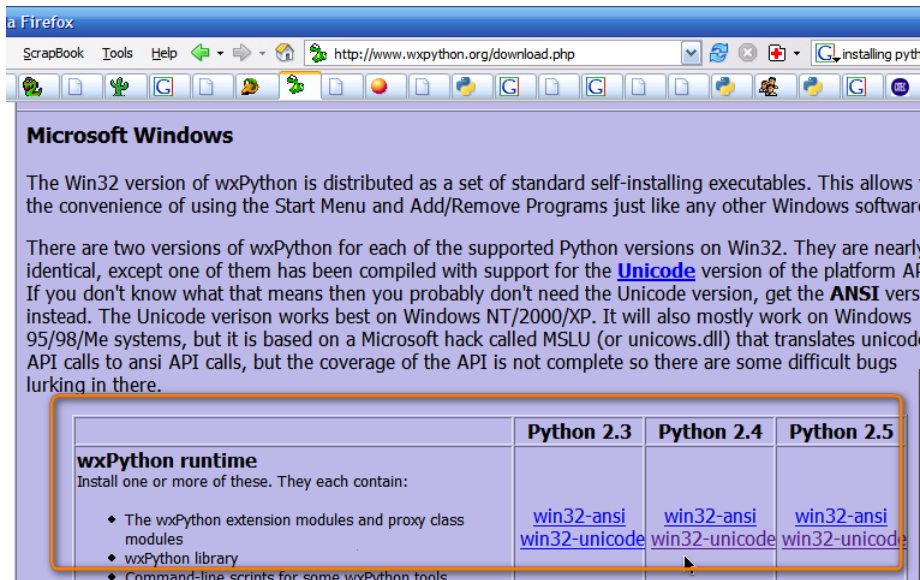


รูปที่ ข.7: ขั้นตอนที่ 6 : เสร็จสิ้นการติดตั้ง

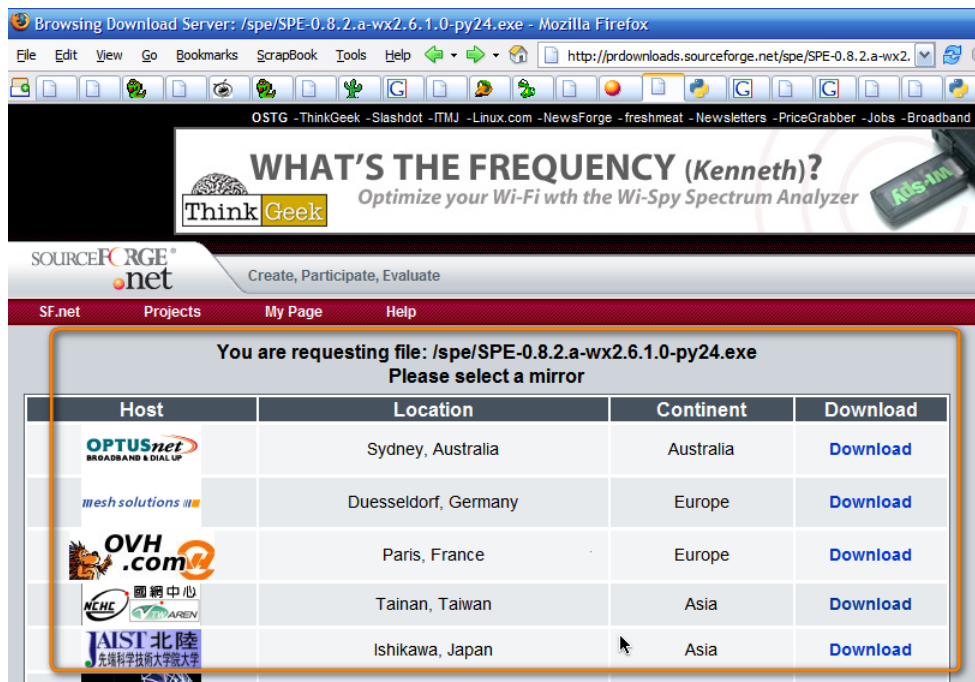
## ข.2 การติดตั้ง wxPython

\* ในเอกสารนี้อ้างอิง wxPython Version 2.6.3.3 win32-unicode for Python 2.4

1. ดาวน์โหลด wxPython Version 2.6 สำหรับ Windows ได้จาก <http://www.wxpython.org/download.php> และไปที่ Microsoft Windows และเลือก win32-unicode จากตาราง wxPython runtime โดยเลือกที่คอลลัม Python 2.4 จากรายการดาวน์โหลด โดย win32-unicode สำหรับ Windows NT/2000/XP และ win32-ansi สำหรับ Windows 95/98/Me



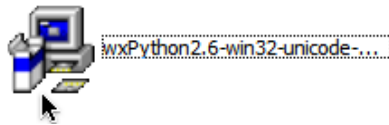
รูปที่ ข.8: เลือกดาวน์โหลด wxPython runtime for Python 2.4



รูปที่ ข.9: เลือกสถานที่ดาวน์โหลด

2. ดับเบิลคลิกที่ wxPython2.6-win32-unicode-2.6.3.3-py24.exe แล้วทำตามขั้นตอนการติดตั้งดังต่อไปนี้

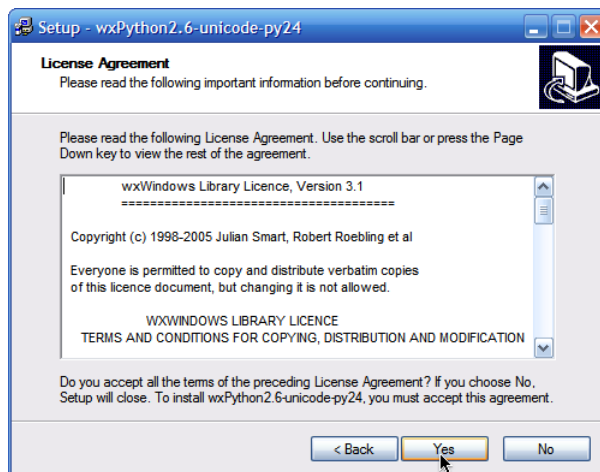




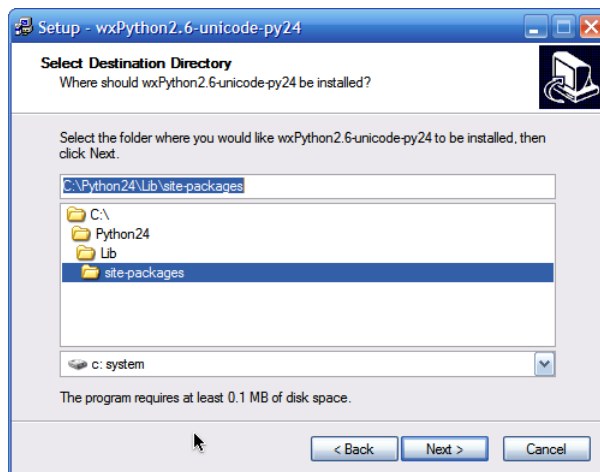
รูปที่ ข.10: ขั้นตอนที่ 1 : ดับเบิลคลิกที่ไฟล์ติดตั้ง



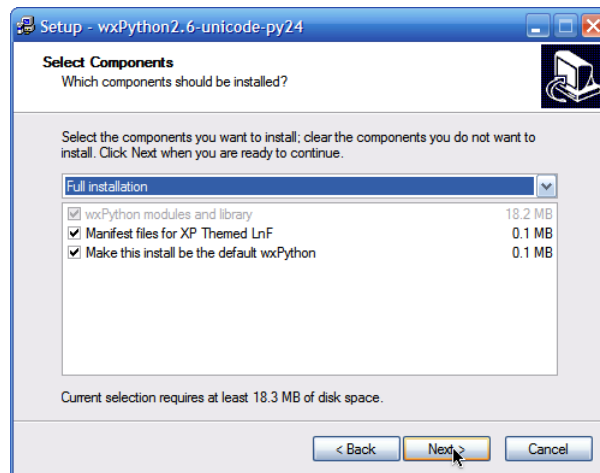
รูปที่ ข.11: ขั้นตอนที่ 2 : หน้าต้อนรับการติดตั้งให้ กด Next



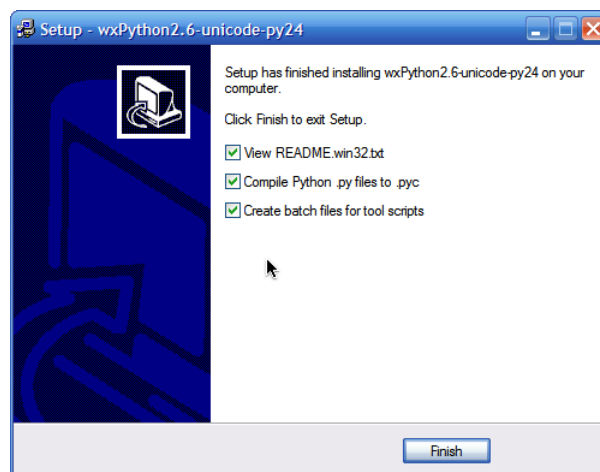
รูปที่ ข.12: ขั้นตอนที่ 3 : หน้ายอมรับข้อตกลงให้ กด Yes



รูปที่ ข.13: ขั้นตอนที่ 4 : หน้าเลือกสถานที่ติดตั้ง ให้เลือกตามที่โปรแกรมได้กำหนดไว้แต่แรก



รูปที่ ข.14: ขั้นตอนที่ 5 : หน้าเลือก component ให้เลือกทั้งหมด แล้วกด Next

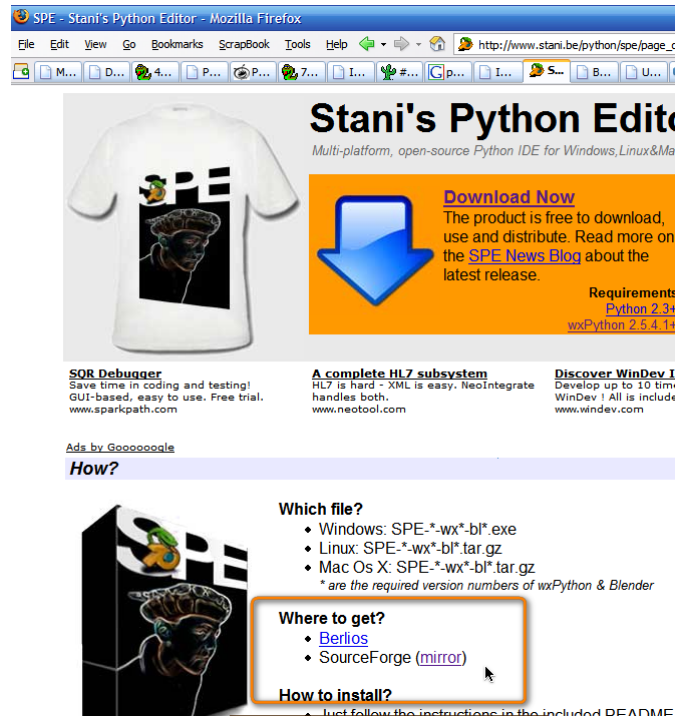


รูปที่ ข.15: ขั้นตอนที่ 6 : เข้าสู่ขั้นตอนการติดตั้งและเมื่อเสร็จแล้วให้เลือก checkbox ทั้งหมดเพื่อให้โปรแกรมติดตั้งทำการดัดแปลงระบบเพิ่มเติมแล้วกด Finish

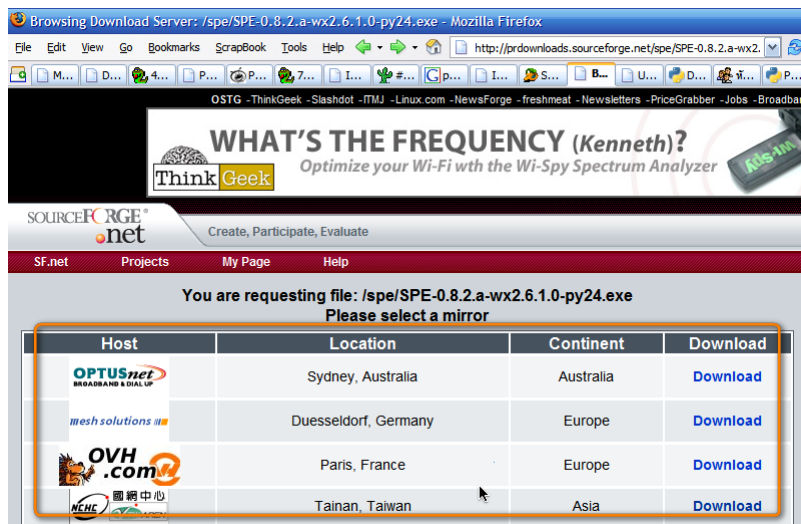
## ข.3 การติดตั้ง Stani's Python Editor

\* ในเอกสารนี้อ้างอิง Stani's Python Editor Version 0.8.2.a for wxPython Version 2.6 win32-unicode and Python 2.4

1. ดาวน์โหลด Stani's Python Editor Version 0.8.2.a ได้จาก [http://www.stani.be/python/spe/page\\_download](http://www.stani.be/python/spe/page_download) และไปที่ Where to get? และเลือก SourceForge (mirror)

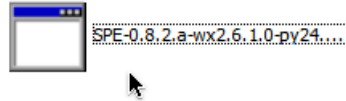


รูปที่ ข.16: ดาวน์โหลด Stani's Python Editor Version 0.8.2.a สำหรับ Windows ได้จากลิงค์ SourceForge (mirror)

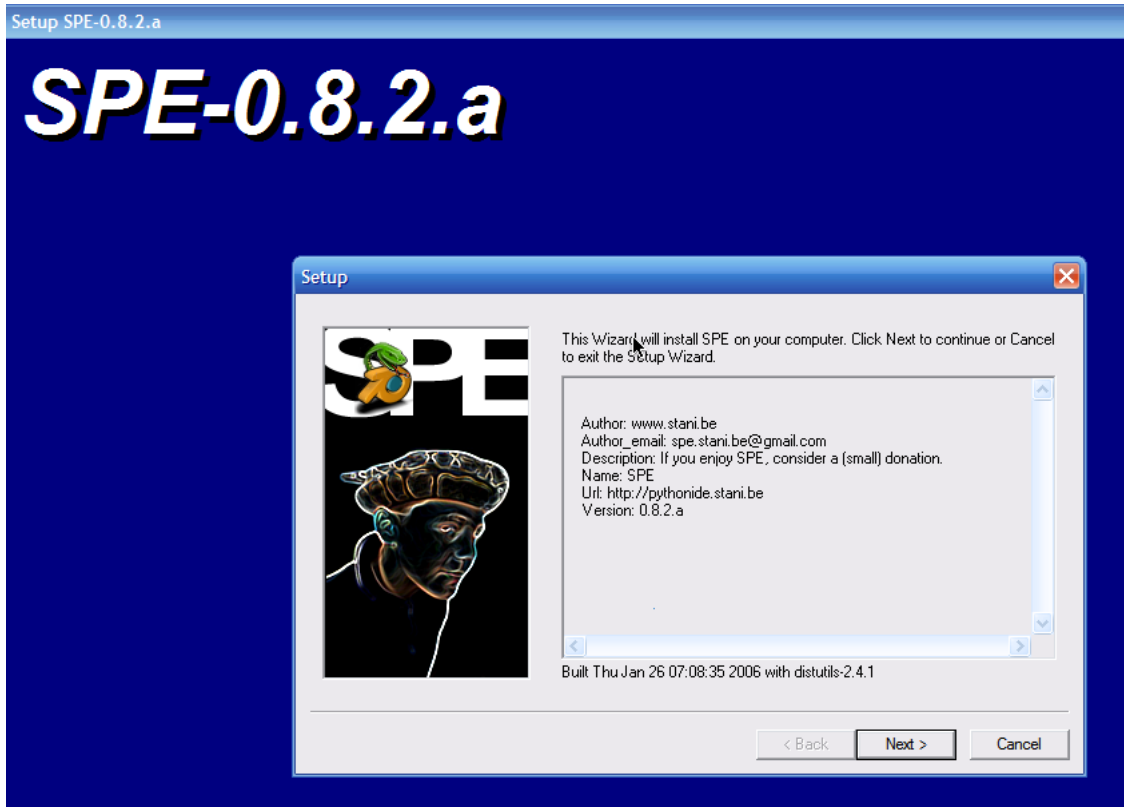


รูปที่ ข.17: เลือกสถานที่ที่ดาวน์โหลด

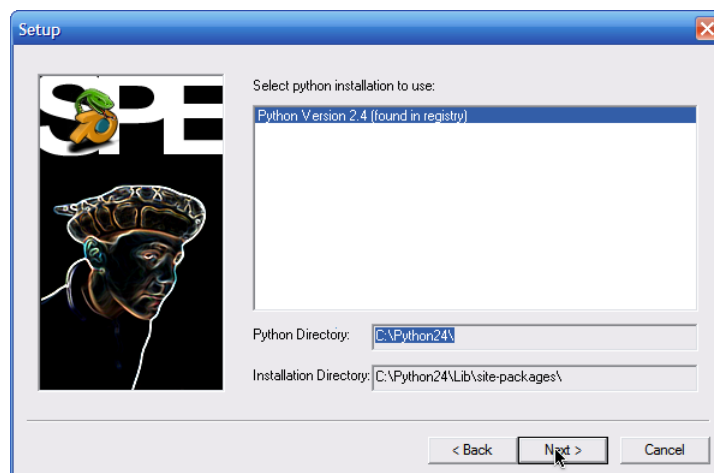
2. ดับเบิลคลิกที่ SPE-0.8.2.a-wx2.6.1.0-py24.exe ที่เป็นไฟล์ที่เราดาวน์โหลด แล้วทำตามขั้นตอนการติดตั้งดังต่อไปนี้



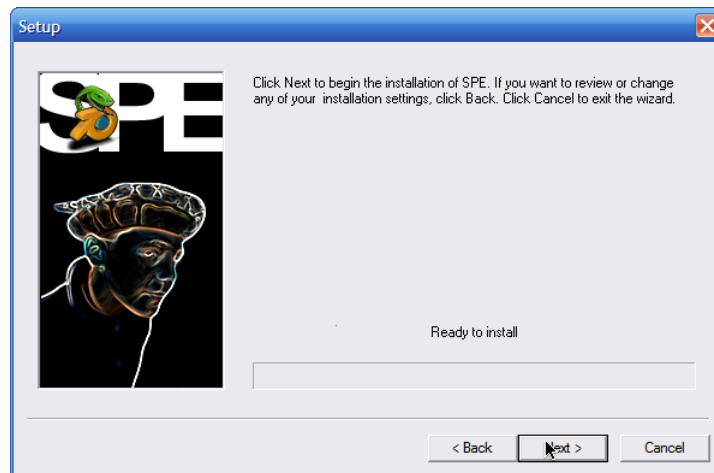
รูปที่ ข.18: ขั้นตอนที่ 1 : ดับเบิลคลิกที่ไฟล์ติดตั้ง



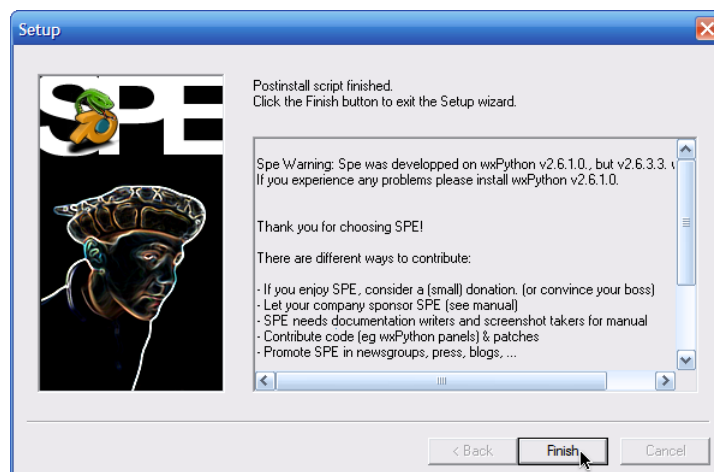
รูปที่ ข.19: ขั้นตอนที่ 2 : หน้าต้อนรับการติดตั้งให้ กด Next



รูปที่ ข.20: ขั้นตอนที่ 3 : หน้าเลือกสถานที่ติดตั้ง ให้เลือกตามที่โปรแกรมได้กำหนดไว้แต่แรก



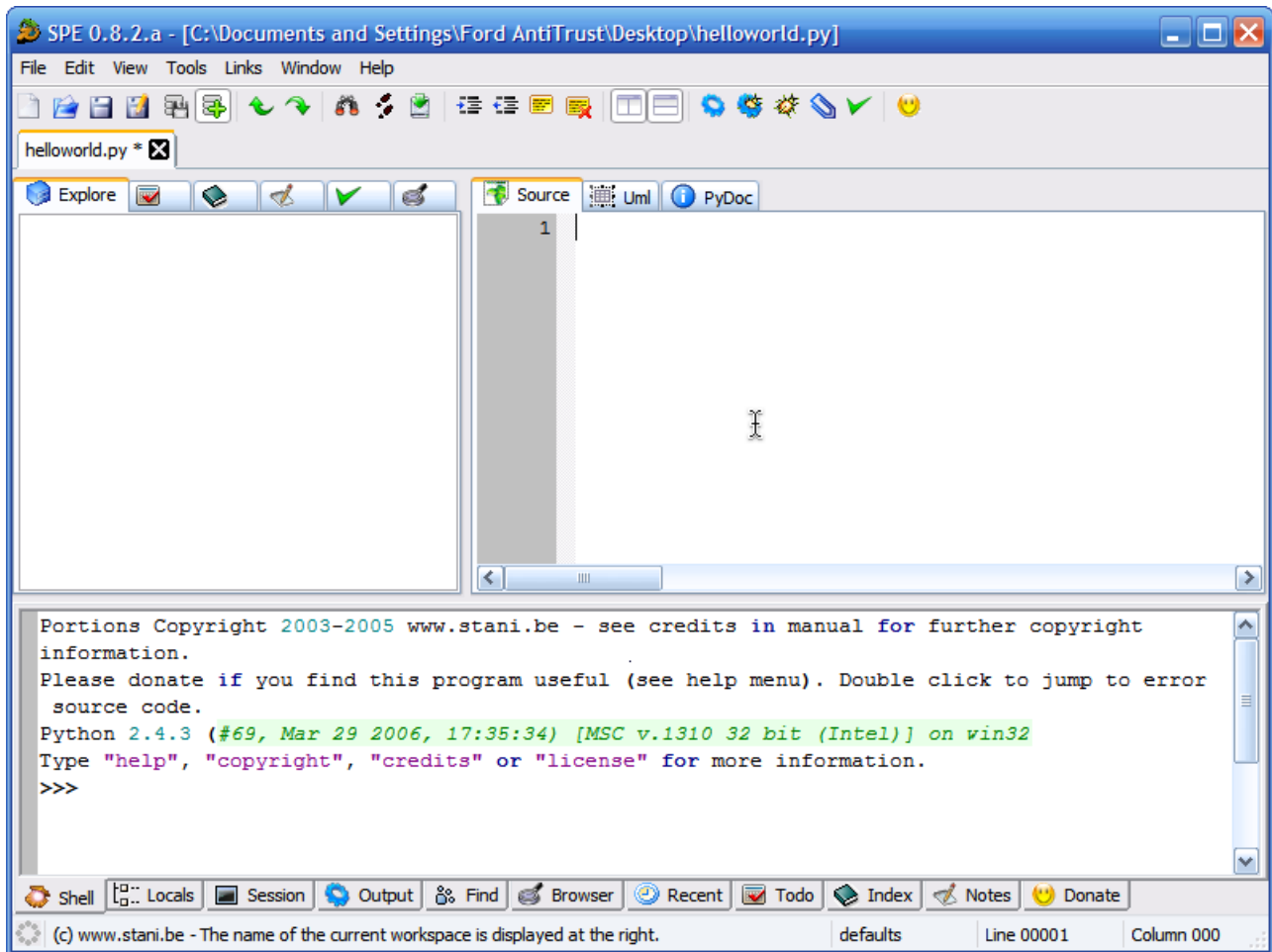
รูปที่ ข.21: ขั้นตอนที่ 4 : เข้าสู่ขั้นตอนการติดตั้งโดยกด Next เพื่อเริ่มการติดตั้ง



รูปที่ ข.22: ขั้นตอนที่ 5 : เข้าสู่ขั้นตอนการติดตั้ง และเสร็จสิ้นการติดตั้ง







รูปที่ ค.2: หน้าต่างโปรแกรม Stani's Python Editor (SPE)

## ค.0.1 Sidebar

### Explore

เป็นหน้าต่าง Class browser สำหรับแสดงรายการ class, method และ function ที่แสดงออกมาเป็นโครงสร้างต้นไม้

### Index

หน้าต่างที่ทำหน้าที่ค้นหาอัตโนมัติสำหรับสร้าง index เพื่อทำ index ให้ง่ายกับการค้นโดยทำ index จาก class และ and method

### Todo

หน้าต่างแสดงรายการที่ต้องทำในไฟล์นั้น ๆ

### Notes

หน้าต่างแสดงหมายเหตุต่าง



## Browser

หน้าต่างในการแสดงไฟล์ใน folder หรือ directory ที่ไฟล์นั้นอยู่ ทำงานคล้าย ๆ กับ Windows Explorer

## ค.0.2 Source

### Editor

หน้าต่างสำหรับการพิมพ์โค้ดคำสั่งต่าง ๆ

### UML view

หน้าต่างแสดง Graphical layout ของลำดับของ class hierarchy

### Pydoc

หน้าต่างสร้างไฟล์ documentation แบบอัตโนมัติ

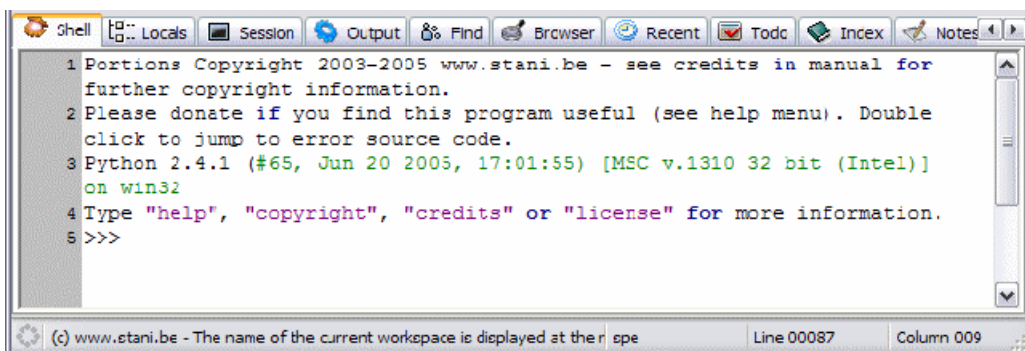
### Separators

หน้าต่างสำหรับใช้ separator เพื่อทำ Label ของชุดคำสั่งของโปรแกรม

## ค.0.3 Tools

### Shell

เป็นส่วนรายงานความผิดพลาดต่าง ๆ ของโปรแกรมที่เราเขียน และสามารถใช้เป็นตัวกระโดดข้ามไปยัง Error line ได้ง่าย และยังใช้ในการพิมพ์คำสั่ง หรือโค้ดโปรแกรมต่าง ๆ แทน Editor ได้ด้วย



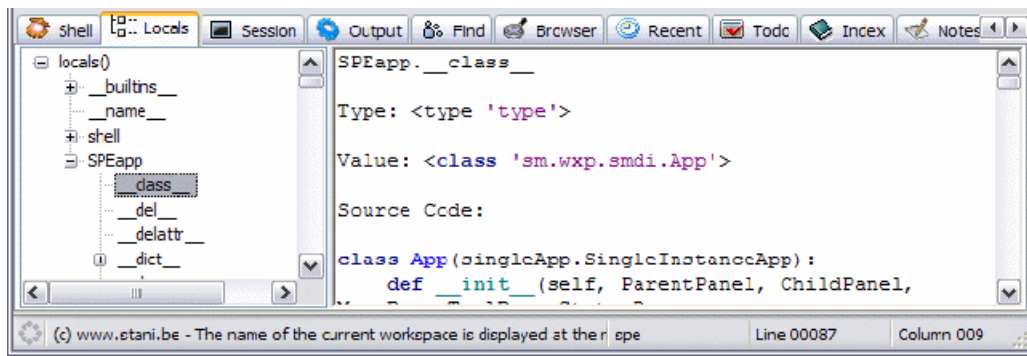
```

Shell | Locals | Session | Output | Find | Browser | Recent | Todo | Index | Notes
1 Portions Copyright 2003-2005 www.stani.be - see credits in manual for
  further copyright information.
2 Please donate if you find this program useful (see help menu). Double
  click to jump to error source code.
3 Python 2.4.1 (#65, Jun 20 2005, 17:01:55) [MSC v.1310 32 bit (Intel)]
  on win32
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>>
(c) www.stani.be - The name of the current workspace is displayed at the r... spe | Line 00087 | Column 009
  
```

รูปที่ ค.3: Shell

### Local object browser

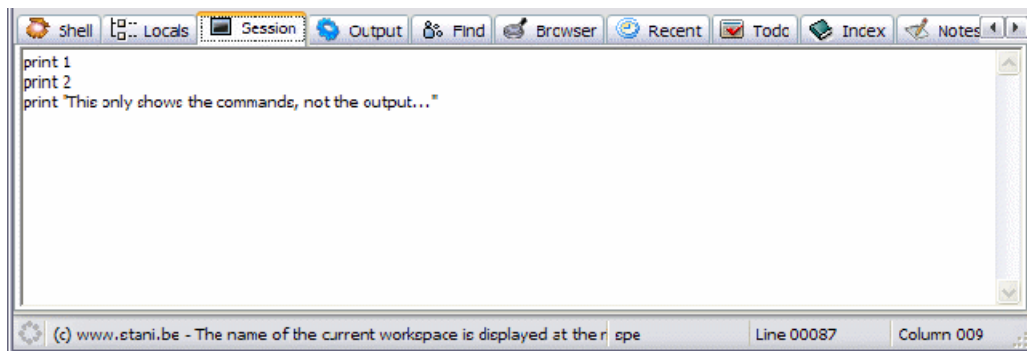
เป็นส่วนบอกรายการตัวแปร, อ็อบเจ็กต์ และคลาส ต่าง ๆ เพื่อใช้ในการดูช่วงของชีวิตของตัวแปร และ อ็อบเจ็กต์ต่าง ๆ และยังช่วยในการค้นหาและกระโดดไปยังตัวแปร, อ็อบเจ็กต์ และคลาส ที่เราต้องการได้ง่าย



รูปที่ ค.4: Local object browser

## Session

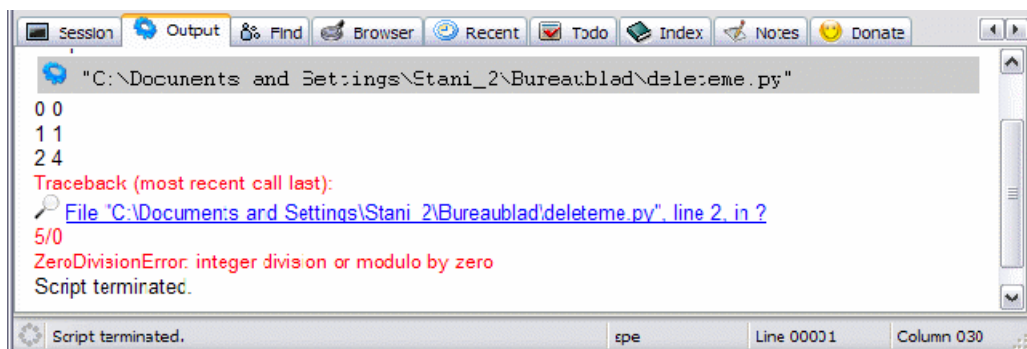
เป็นตัวบันทึกการพิมพ์โค้ดโปรแกรมสำหรับช่วยในการไม่ต้องพิมพ์ใหม่ในกรณีที่ใช้ shell ในการพิมพ์คำสั่งโปรแกรมต่าง ๆ



รูปที่ ค.5: Session

## Output

ถ้าคุณสั่งให้โปรแกรมทำงานใน SPE โดยใช้คำสั่ง Tools > Run/Stop ตัวผลการทำงานจะออกมาในหน้าต่างนี้ รวมถึง Error ต่าง ๆ โดยสามารถกระโดดไปยังโค้ดหรือชุดคำสั่งที่ Error นั้น ๆ ได้จากหน้าต่างนี้



รูปที่ ค.6: Output

## อื่น ๆ

1. Find หน้าต่างสำหรับการค้นหาข้อมูลต่าง ๆ
2. Browser หน้าต่างที่ใช้ในการเรียกดูชื่อไฟล์ซึ่งทำงานเหมือน Windows Explorer
3. Recent หน้าต่างแสดงรายชื่อไฟล์ที่เราเปิดอยู่
4. Todo หน้าต่างแสดงรายการที่ต้องทำใน Project หรือไฟล์ที่เราเปิดอยู่
5. Index หน้าต่างคำสั่งสำหรับเรียกใช้สารบัญงานต่าง ๆ ภายในไฟล์
6. Notes หน้าต่างคำสั่งหมายเหตุสำหรับป้องกันการหลงลืม



## ภาคผนวก ง

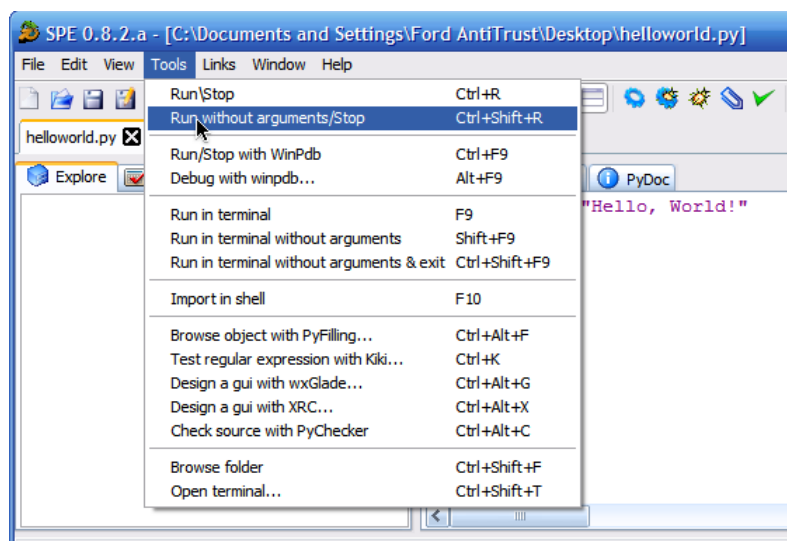
# การเขียน, Debug และสั่งให้โปรแกรมทำงาน

### ง.1 การเขียนโปรแกรมใน SPE และสั่งให้โปรแกรมทำงาน

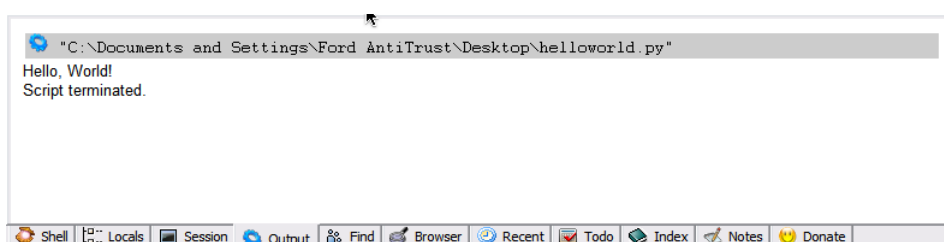
เพื่อทดสอบการทำงานให้พิมพ์คำสั่งโปรแกรมเริ่มต้นก่อนโดยพิมพ์คำสั่งดังนี้

```
>>> print "Hello, World!"
```

พิมพ์ลงใน Source Editor แล้วไปที่ "Tools" ที่ menu bar แล้วไปที่ คำสั่ง "**Run without arguments/Stop**" หรือกด short-key *Ctrl+Shift+R*

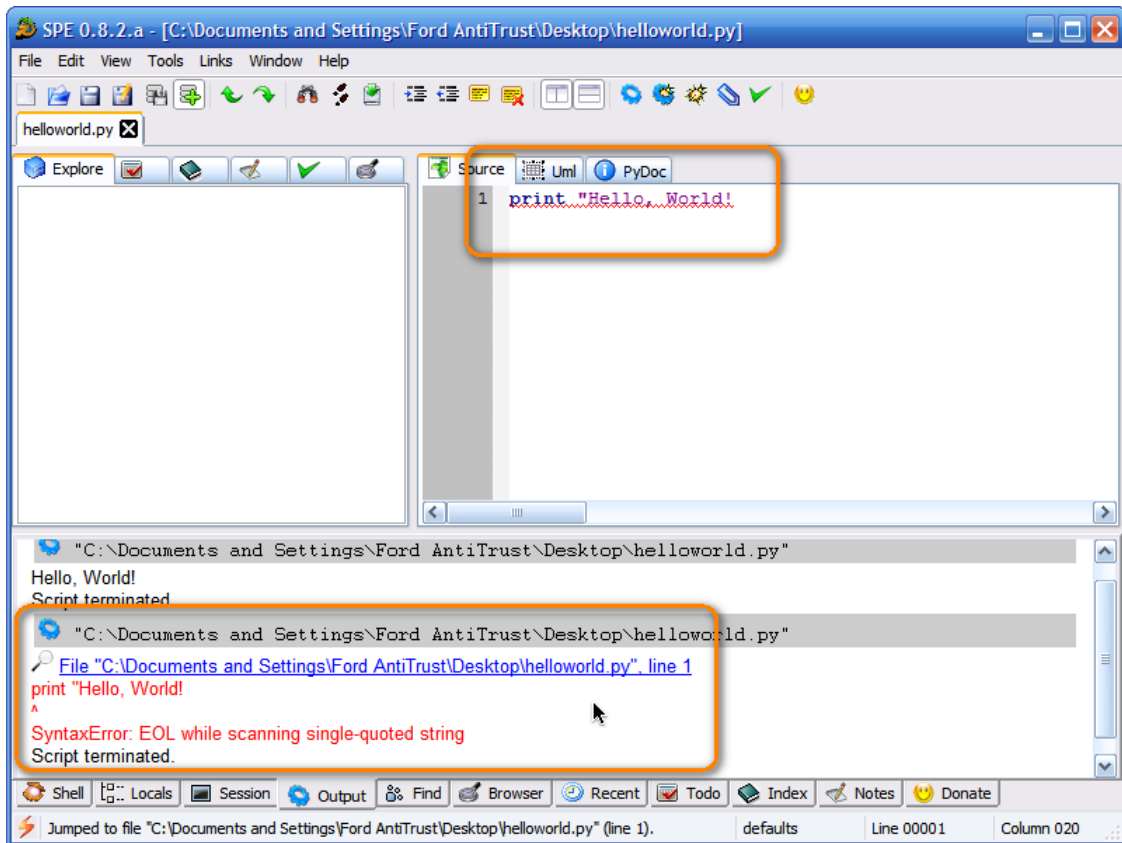


รูปที่ ง.1: สั่งให้โปรแกรมทำงาน



รูปที่ ง.2: ผลการทำงาน

## ง.2 การ Debug โปรแกรม



รูปที่ ง.3: การแจ้ง Error เพื่อใช้ประกอบการ Debug

```
File "C:\Documents and Settings\Ford AntiTrust\Desktop\helloworld.py", line
print "Hello, World!"
^
```

```
SyntaxError: EOL while scanning single-quoted string
```

จากตัวอย่างได้แจ้ง Error ได้สองที่ด้วยกัน คือหน้าต่าง Editor ซึ่งจะมีเส้นใต้เป็นฟันทาสีแดงขีดตามบรรทัดที่มี Error และในหน้าต่าง Output ก็มี Error Code แจ้งไว้ โดยได้บอกไว้ว่าที่ line 1 มี Error โดยบอกว่าเป็น SyntaxError และมีปัญหาจาก Systax ลักษณะ single-quoted string มีปัญหานั้นเอง

โดยปัญหาในการแจ้งต่าง ๆ นั้นตัว Interpreter จะแจ้งให้ทราบได้จาก line ที่มีปัญหา เช่นตัวอย่างด้านล่างนี้ได้แจ้งไว้ว่ามาจาก line ที่ 1

```
File "C:\Documents and Settings\Ford AntiTrust\Desktop\helloworld.py", line
```

รวมไปถึงแจ้ง column ของ line ว่าน่าจะผิดที่คำสั่งใด ในที่นี้ผิดที่คำสั่ง print นั้นเอง

```
print "Hello, World!"
```

```
^
```

```
SyntaxError: EOL while scanning single-quoted string
```

## ภาคผนวก จ

### ข้อมูลอ้างอิง

- *Python programming language*. <http://www.python.org/>, Python Software Foundation, 2006.
- *Python programming language*. [http://en.wikipedia.org/wiki/Python\\_programming\\_language](http://en.wikipedia.org/wiki/Python_programming_language), Wikipedia the free encyclopedia, 2006.
- *Python software*. [http://en.wikipedia.org/wiki/Python\\_software](http://en.wikipedia.org/wiki/Python_software), Wikipedia the free encyclopedia, 2006.
- *Comparison of programming languages*. [http://en.wikipedia.org/wiki/Comparison\\_of\\_programming\\_languages](http://en.wikipedia.org/wiki/Comparison_of_programming_languages), Wikipedia the free encyclopedia, 2006.
- *Python Tutorial Online*. <http://python.cmsthailand.com/>, CMSthailand.com, 2005.
- *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl — March 2000 refereed journal paper*. [http://page.mi.fu-berlin.de/~prechelt/Biblio/jccpprt\\_computer2000.pdf](http://page.mi.fu-berlin.de/~prechelt/Biblio/jccpprt_computer2000.pdf), Lutz Prechelt, 2000.
- *Programming in Python*. <http://www-128.ibm.com/developerworks/library/os-python5/index.html>, Robert Brunner (rb@ncsa.uiuc.edu), Research Scientist, National Center for Supercomputing Applications, 2005.
- *wxPython, a blending of the wxWidgets C++ class library*. <http://www.wxpython.org/>, OS-AF (Open Source Applications Foundation), 2006.
- *Stani's Python Editor, Python IDE with Blender, Kiki, PyChecker, wxGlade & XRC support*. <http://pythonide.stani.be/>, S. Michiels, Amsterdam, the Netherlands 2006.
- *Beginning Python* Wiley Publishing, Inc., Peter Norton, Alex Samuel, David Aitel, Eric Foster-Johnson, Leonard Richardson, Jason Diamond, Aleatha Parker, Michael Roberts, 2005.
- *Python* <http://veer.exteen.com/>, วีร์ สัตยมาศ, 2005.
- *เรื่องที่ทำมึนใน Python* <http://plynoi.exteen.com/20060827/python>, Plynoi, 2006.

# ดรรชนี

- 64 bit double precision, 30
- Alternatively, 50
- Category และ Application Domains, 16
  - 3D Graphics Rendering, 17
  - Database Access, 16
  - Desktop GUI, 17
  - Education, 17
  - Game, 17
  - GUI, 17
  - Network programming, 17
  - Numeric computation, 17
  - Scientific, 17
  - Software builder, 17
  - Software Testing, 17
  - Web และ Internet Development, 16
- Colon Symbol, 32
- Comment, 22
- Concatenation Symbol, 40
- Control flow, 50
- Data type, 27
  - ชนิดข้อมูลแบบการรวมกลุ่มข้อมูล (Collection Data Types), 32
  - ดิกชันนารี (Dictionary หรือ Groupings of Data Indexed by Name), 33
  - ทับเปิ้ล (Tuples) และ อนุกรม (Sequences), 33
  - ลิสต์ (List), 32
  - อาร์เรย์ (Array), 32
  - เซต (Sets), 34
  - ตัวเลข (Numbers), 29
    - imaginary number, 31
    - signed integer, 29
    - จำนวนตรรกะ (Boolean), 30
    - จำนวนจริง (Floating-point numbers), 30
    - จำนวนเชิงซ้อน (Complex Numbers), 31
    - จำนวนเต็ม (Integer), 29
    - จำนวนเต็มธรรมดา (Plain Integer), 29
    - จำนวนเต็มแบบยาว (Long integer), 30
    - สายอักขระ (String หรือ Array of Characters), 39
- Flow of Control, 50
- Function argument, 59
- Functions
  - abs(), 26
  - complex(), 31
  - float(), 31
  - int(), 30
  - len(), 34
  - max(), 26
  - min(), 26, 27
  - print, 21
  - range(), 27, 53–55
  - raw\_input, 63
  - set(), 35
  - sum(), 27
  - type(), 29
- Google, 19
- Guido van Rossum, 13
- Hello World, 21, 85
- Industrial Light & Magic, 18
- Language Evaluation Criteria, 15
  - Cost, 15
  - Readability, 15
  - Reliability, 15
  - Writability, 15
- Multi-paradigm language, 15



NASA, 19

Operator

\*, 25

\*\*, 25, 30

+, 25

-, 25

/, 25

%, 25

Python, 13

Repetition Symbol, 21

Reserved words (Keywords), 22

and, 22, 47

as, 22

assert, 22, 55

break, 22, 54

class, 22

continue, 22, 54

def, 22, 58

del, 22

elif, 22, 51

else, 22, 51

else clauses, 54

else if, 51

except, 22, 56

exec, 22

finally, 22

for, 22, 53

from, 22

global, 22, 49, 62

if, 22, 51

import, 22

in, 22, 45, 53–55

is, 22, 45

lambda, 22

not, 22, 47

or, 22, 47

pass, 22, 54

print, 22

raise, 22, 56

return, 59

return , 22

try, 22, 56

while, 22, 53

with, 22

yield, 22

Stani's Python Editor (SPE), 75

Browser, 81

Explore, 80

Index, 80

Notes, 80

Source, 81

Editor, 81

Pydoc, 81

Separators, 81

UML view, 81

Todo, 80

Tools, 81

Local object browser, 81

Output, 82

Session, 82

Shell, 81

อื่น ๆ, 83

การเขียน, Debug และสั่งให้โปรแกรมทำงาน, 85

การ Debug โปรแกรม, 86

การสั่งให้โปรแกรมทำงาน, 85

การเขียนโปรแกรมใน SPE, 85

หน้าต่างโปรแกรม, 80

statement block, statement scope, 48

String Concatenation Symbol, 40

String formatting operator, 41

Symbol

\*, 21

+, 40

„, 40

:, 32

#, 22

% (String formatting operator), 41

type

bool, 30

complex, 31

dict, 33

- False, 30
- float, 30
- int, 29
- list, 32
- long, 30
- set, 34
- True, 30
- tuple, 34
- wxPython, 72
- การควบคุมทิศทางของโปรแกรม, 50
  - การจัดการความผิดปกติของโปรแกรม (Error Checking), 51, 55
  - assert Statements, 55
  - try-except และ raise Statements (Exception handling), 56
  - การตัดสินใจ (Decisions, Choice or Selection), 51
  - การตัดสินใจ (Decisions, Choice หรือ Selection)
    - elif, 51
    - else, 51
    - else if, 51
    - if, 51
    - switch, 52
  - การวนซ้ำ (Loop, Iteration), 51
    - break, 54
    - continue, 54
    - do-while, 55
    - else clauses, 54
    - for, 53
    - while, 53
- การคำนวณทางคณิตศาสตร์ (Arithmetic Mathematics), 23
  - การคำนวณผ่านฟังก์ชันภายใน (Built-in Math Functions), 26
  - การหาค่าสัมบูรณ์, 26
  - กำหนดจำนวนตัวเลขทศนิยม, 27
  - จำนวนที่น้อยที่สุด และมากที่สุดในกลุ่ม, 26
  - ช่วงของข้อมูลตัวเลข, 27
  - หาผลรวมทั้งหมดในชุดข้อมูล, 27
  - การคำนวณพื้นฐาน (normal arithmetic operators), 25
    - คูณ, 25
    - บวก, 25
    - ยกกำลัง, 25
    - ลบ, 25
    - หาร, 25
    - หารเอาเศษ, 25
    - อันดับความสำคัญของการคำนวณ, 25
  - การตั้งตัวชื่อแปร, 22
  - การติดตั้ง, 69
    - Python, 69
    - Stani's Python Editor (SPE), 69, 75
    - wxPython, 69, 72
  - การสร้างฟังก์ชัน (Defined Function), 58
    - Global Variable, 62
    - Local Variable, 62
    - การคืนค่ากลับ (return) , 60
    - การรับค่าของฟังก์ชัน (Function Argument), 60
    - ค่ามาตรฐานของการรับค่า (Function Default Argument), 60
  - การเปรียบเทียบ (Comparisons), 43
    - ตัวอย่าง, 45
  - การแสดงผลเบื้องต้น (Printing) และสัญลักษณ์ที่เกี่ยวข้อง, 19
  - ข้อเด่นของภาษาไพธอน, 15
  - คำสงวน, 22
  - ชนิดของตัวแปร, 27
  - ช่วงของการทำงาน, 48
  - ช่วงของการทำงาน (Statement block), 49
  - ช่วงชีวิตของตัวแปร (Life time หรือ Variable scope), 49
  - ซอฟต์แวร์ที่เขียนด้วยไพธอน, 17
    - Battlefield 2, 18
    - BitTorrent, 17
    - Blender, 18
    - Chandler, 18
    - Civilization IV, 18
    - EVE Online, 18
    - Indian Ocean Tsunami Detector, 18

Kombilo, 18  
Mailman, 18  
MoinMoin, 18  
OpenRPG, 18  
Plone, 18  
Stani's Python Editor(SPE), 18  
Trac, 18  
Turbogears, 18  
ViewVC, 18  
Zope, 18

นิพจน์ทางตรรกศาสตร์ (Boolean Expressions), 46

AND, 47  
NOT, 47  
OR, 47

ประวัติ, 13

Python 1.0, 13  
Python 2.0, 14  
อนาคต, 14

หลักปรัชญาของภาษาไพธอน, 15