P Y T H O N

ใคร ๆ ก็เขียนโปรแกรมได้

ประเสริฐ คณาวัฒนไชย

Python

ใคร ๆ ก็เขียนโปรแกรมได้

ผู้แต่ง **ประเสริฐ คณาวัฒนไชย**

พิมพ์ครั้งที่ 1

ISBN e-book 978-616-608-212-8

ราคา 279 บาท

จัดทำโดย ประเสริฐ คณาวัฒนไชย

สั่งซื้อได้ที่ ศูนย์หนังสือจุฬาลงกรณ์มหาวิทยาลัย
ถนนพญาไท เขตปทุมวัน กรุงเทพฯ 10330
โทร. 0-2218-9872 โทรสาร 0-2254-9495
Call Center โทร. 0-2255-4433
http://www.chulabook.com

ดาวน์โหลด source code ที่ใช้ในตำราได้ที่: https://github.com/prasertcbs/pybook

ข้อคิดเห็นหรือข้อเสนอแนะ สามารถส่งมาได้ที่อีเมล prasert.k@chula.ac.th

คำนำ

Steve Jobs หนึ่งในผู้ก่อตั้งบริษัท Apple เคยกล่าวไว้ในปี 1995 ว่า

“Everyone in this country should learn how to program because it teaches you how to think.”

คำกล่าวนี้สะท้อนให้เห็นถึงความสำคัญอย่างยิ่งของการเรียนรู้เกี่ยวกับเขียนโปรแกรมคอมพิวเตอร์ เพราะมันจะช่วยให้เราหัดคิดและแก้ปัญหาอย่างเป็นขั้นตอน และเป็นหนึ่งในทักษะที่ควรมีในโลกยุคปัจจุบัน

ตำราเล่มนี้จะพาคุณไปพบกับการเขียนโปรแกรมด้วยภาษาไพทอน (Python) ซึ่งเป็นภาษาที่ได้รับความนิยมอย่างแพร่หลายทั้งในภาคอุตสาหกรรม ธุรกิจ และการศึกษา เนื่องจากเป็นภาษาที่มีโครงสร้างและชุดคำสั่งที่เรียบง่าย ทำให้อ่านและทำความเข้าใจได้ง่าย เหมาะสำหรับผู้หัดเรียนเขียนโปรแกรม และนำไปต่อยอดในด้านต่าง ๆ ได้อย่างมากมาย

เนื้อหาในตำราเล่มนี้ถูกกลั่นกรองมาจากประสบการณ์ในการเป็นนักพัฒนาและการเป็นผู้สอนการเขียนโปรแกรมกว่า 30 ปี โดยเน้นให้เข้าใจง่ายสำหรับผู้เริ่มต้นที่ไม่มีพื้นฐานเริ่มต้นในการเขียนโปรแกรม เริ่มตั้งแต่แนวคิดและหลักการทำงานของคอมพิวเตอร์ ภาษาคอมพิวเตอร์ ตัวแปร การควบคุมการทำงานของโปรแกรม โครงสร้างข้อมูลพื้นฐาน ฟังก์ชัน การอ่านเขียนไฟล์ข้อมูล และการจัดการข้อผิดพลาดที่อาจเกิดขึ้น โดยเนื้อหาในแต่ละส่วนจะเน้นให้เห็นถึงการประยุกต์ใช้คำสั่งต่าง ๆ ผ่านตัวอย่างที่เราพบเห็นในชีวิตประจำวัน เพื่อให้เป็นแนวทางในการนำไปปรับใช้งานในบริบทต่าง ๆ ได้

ถึงแม้หลายคนจะมองการเขียนโปรแกรมเป็นเรื่องยาก และต้องมีความสามารถพิเศษถึงจะเขียนได้ แต่ผู้เขียนมั่นใจว่า “**การเขียนโปรแกรมคือศิลปะของการคิด**” ทุกคนมีศักยภาพที่จะสามารถเขียนโปรแกรมได้ หากมีความตั้งใจและมุ่งมั่น ผู้เขียนอยากเห็นหนังสือเล่มนี้เป็นจุดเริ่มต้นในการจุดประกายความอยากรู้อยากเห็นในการเขียนโปรแกรมคอมพิวเตอร์ เพื่อนำไปต่อยอดในงานด้านต่าง ๆ ได้ในภายหน้า โดยเฉพาะการให้ผู้อ่านได้เห็นถึงศักยภาพและความเป็นไปได้ของการนำเอาคอมพิวเตอร์ไปประยุกต์ใช้ในงานด้านต่าง ๆ ไม่ว่าจะเป็นแค่งานอดิเรก การเรียน จนถึงการนำไปใช้ในการทำงาน

มาร่วมกันเปิดประสบการณ์ใหม่ ๆ และสนุกไปกับการหัดเขียนโปรแกรมคอมพิวเตอร์ด้วยภาษาไพทอนกันครับ

รศ. ดร. ประเสริฐ คณาวัฒนไชย
คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย
พฤศจิกายน 2566

สารบัญย่อ

สารบัญ

1. ทำไมต้องเขียนโค้ด

วัตถุประสงค์การเรียนรู้

หลังจากศึกษาเนื้อหาในบทนี้แล้วผู้อ่านควรที่จะ

- รู้ว่าโปรแกรมคอมพิวเตอร์คืออะไร ทำงานอย่างไร
- อธิบายขั้นตอนการทำงานของโปรแกรมคอมพิวเตอร์ได้
- รู้ว่าภาษาคอมพิวเตอร์แบ่งออกได้เป็นกี่ประเภท
- อธิบายได้ว่า Interpreter กับ Compiler ต่างกันอย่างไร พร้อมข้อดีข้อเสีย และการนำไปใช้งาน

1.1 โปรแกรมคอมพิวเตอร์คืออะไร

โปรแกรมคอมพิวเตอร์ (computer program) เป็นลำดับของคำสั่งที่ออกแบบมาเพื่อดำเนินการงานที่เจาะจงบนเครื่องคอมพิวเตอร์ คำสั่งเหล่านี้ถูกสร้างขึ้นโดยใช้ภาษาที่คอมพิวเตอร์สามารถแปลความหมายได้ อาทิ C, C++, Java, Python, และ JavaScript

หัวใจหลักในการเขียนโปรแกรมคอมพิวเตอร์ คือ การวิเคราะห์และทำความเข้าใจปัญหาที่ต้องการแก้ไข จากนั้นจึงนำมาออกแบบโครงสร้างของโปรแกรม ตามด้วยการเขียนโค้ด ซึ่งแต่ละภาษาจะมีไวยากรณ์และชุดคำสั่งที่แตกต่างกัน ผู้พัฒนาซอฟต์แวร์จะเลือกใช้ภาษาและเครื่องมือพัฒนาที่เหมาะสมเพื่อการเขียนและทดสอบโปรแกรม จนกว่าโปรแกรมจะสามารถทำงานได้ตามข้อกำหนดที่ได้ระบุไว้ รวมถึงการจัดทำคู่มือและเอกสารการใช้งาน ตัวอย่างของโปรแกรมคอมพิวเตอร์ที่พบเห็นได้ทั่วไป เช่น เว็บเบราว์เซอร์ โปรแกรมการคำนวณราคาสินค้าในร้านสะดวกซื้อ, โปรแกรมการรับจ่ายเงินด้วยระบบ PromptPay, โปรแกรมรับส่งอีเมล

1.2 ทำไมถึงควรเรียนรู้การเขียนโปรแกรม

การเขียนโปรแกรมเป็นจะช่วยให้เราทำงานต่าง ๆ ได้อย่างรวดเร็วและมีประสิทธิภาพ ประโยชน์ที่ได้รับอาจสรุปเป็นข้อ ๆ ได้ดังนี้

1. การทำงานอย่างอัตโนมัติหรือกึ่งอัตโนมัติ: การเขียนโปรแกรมช่วยให้เราสามารถทำงานที่ซ้ำซ้อนได้อย่างอัตโนมัติ ทำให้เราสามารถประหยัดเวลาและเพิ่มประสิทธิภาพในการทำงานส่วนตัว และการดำเนินธุรกิจของได้ เช่น โปรแกรมสำหรับสรุปยอดขายสินค้าแต่ละประเภท แยกตามสาขาตามวัน

2. การประมวลผลข้อมูล: การประมวลผลข้อมูลเป็นส่วนสำคัญในการทำงานด้านต่าง ๆ เราสามารถเขียนโปรแกรมเพื่อช่วยให้เราสามารถทำความเข้าใจและวิเคราะห์ข้อมูลได้อย่างมีประสิทธิภาพ และช่วยในการตัดสินใจทางธุรกิจที่มีข้อมูลเป็นพื้นฐาน เช่น การใช้ AI มาช่วยในการวิเคราะห์ข้อมูลต่าง ๆ การพยากรณ์ความต้องการของลูกค้า การวิเคราะห์การให้สินเชื่อ
3. การหัดคิดแก้ปัญหา: การเขียนโปรแกรมเป็นกระบวนการที่สอนให้เราคิดเชิงวิเคราะห์และแก้ปัญหาอย่างเป็นขั้นตอนอย่างมีระบบ
4. การติดต่อสื่อสาร: การเรียนรู้การเขียนโปรแกรมช่วยเสริมสร้างความรู้และทักษะทางด้านเทคโนโลยี ทำให้เราสามารถเข้าใจและสื่อสารกับทีมทำงานที่มีพื้นฐานการเขียนโปรแกรมเพื่อให้พัฒนาโปรแกรมตามที่ต้องการได้ รวมถึงการสื่อการกับเครื่องมือต่าง ๆ เช่น ChatGPT, Google Bard ได้ดียิ่งขึ้น

> คลิปศึกษาเพิ่มเติมบน YouTube
>
> 📺 ทำไมเราถึงควรที่จะเรียนการเขียนโปรแกรมคอมพิวเตอร์

1.3 ภาษาคอมพิวเตอร์คืออะไร

ในโลกใบนี้มีภาษาที่มนุษย์ใช้กันในชีวิตประจำวันอยู่มากมาย ไม่ว่าจะเป็น ภาษาอังกฤษ ภาษาจีน ภาษาไทย ภาษาสเปน ซึ่งภาษาเหล่านี้ถูกสร้างขึ้นโดยมีวัตถุประสงค์หลักในการสื่อสาร เพื่อให้ผู้รับสารมีความเข้าใจถึงเนื้อหาตามที่ผู้ส่งสารต้องการ หากมนุษย์ต้องการติดต่อสื่อสารกับคอมพิวเตอร์จะทำได้อย่างไร คำตอบก็คือ เราจำเป็นต้องเรียนรู้ไวยากรณ์และคำสั่งต่าง ๆ ของภาษาคอมพิวเตอร์ที่เราต้องการใช้ เพื่อจะได้สั่งงานให้คอมพิวเตอร์ทำงานตามที่เราต้องการได้อย่างถูกต้อง

ภาษาคอมพิวเตอร์ (computer language) คือภาษาที่ใช้ในการเขียนโปรแกรมคอมพิวเตอร์ เพื่อให้คอมพิวเตอร์เข้าใจและปฏิบัติตามคำสั่งที่ระบุไว้ โดยในแต่ละภาษาจะมีกฎเกณฑ์และไวยากรณ์ที่ชัดเจนและเฉพาะเจาะจงเพื่อให้คอมพิวเตอร์เข้าใจและทำงานตามที่ผู้ใช้ต้องการ ในปัจจุบันมีภาษาคอมพิวเตอร์จำนวนมากมายนับพันภาษา โดยแต่ละภาษาที่ถูกสร้างขึ้นก็จะมีความสามารถในบางด้านที่แตกต่างกันไป บางภาษาจะเน้นไปทางการพัฒนาเว็บไซต์ เช่น JavaScript, TypeScript, PHP ในขณะที่ภาษาอื่น ๆ อาจเน้นใช้สำหรับการพัฒนาซอฟต์แวร์และแอปพลิเคชันต่าง ๆ เช่น C++, C#, Java, Python, Swift, Kotlin, Go และ Rust หรือบางภาษาเหมาะสำหรับการใช้เขียนระบบปฏิบัติการ เช่น C นอกจากภาษาที่กล่าวไปข้างต้น ก็ยังมีอีกหลายภาษาที่เหมาะกับการนำไปใช้งานที่แตกต่างกันออกไป

ภาษาคอมพิวเตอร์สามารถแบ่งออกเป็นหลายระดับ ได้แก่

1. **ภาษาเครื่อง (Machine Language)** ซึ่งเป็นภาษาที่ใช้คำสั่งที่เครื่องคอมพิวเตอร์เข้าใจได้โดยตรง แต่จะอ่านและเขียนได้ยากมาก เนื่องจากใช้รหัสทางเลขฐานสอง (Binary) ถือเป็น Low-level Language

2. **ภาษาแอสเซมบลี (Assembly Language)** มีคำสั่งที่ใช้ตัวย่อสัญลักษณ์แทนคำสั่งเครื่องคอมพิวเตอร์ เป็นภาษาที่ใกล้เคียงกับภาษาเครื่องแต่มีระดับสูงขึ้น

ลองมาดูตัวอย่างโปรแกรมภาษา Assembly ของชิปตระกูล x86 เพื่อแสดงข้อความ `Hello, World!` ออกทางหน้าจอ

โปรแกรม 1.1: ภาษา Assembly ของชิปตระกูล x86 เพื่อแสดงข้อความ Hello, World! ออกทางหน้าจอ

```
section .data
    hello db 'Hello, World!', 0

section .text
    global _start

_start:
    ; เรียกฟังก์ชัน write
    mov eax, 4            ; system call number (write)
    mov ebx, 1            ; file descriptor (stdout)
    mov ecx, hello        ; ข้อความที่จะแสดง
    mov edx, 13           ; ความยาวข้อความ (13 ตัวอักษร)
    int 0x80              ; เรียกฟังก์ชันเคลื่อนที่ไปยังการแปลงสายอักขระ

    ; เรียกฟังก์ชัน exit
    mov eax, 1            ; system call number (exit)
    xor ebx, ebx          ; ค่าการออกจากระบบ 0 (ไม่มีข้อผิดพลาด)
    int 0x80              ; เรียกฟังก์ชันเคลื่อนที่ไปยังการสิ้นสุดการทำงาน
```

ตัวอย่างถัดไปจะเป็นตัวอย่างโปรแกรมภาษา Assembly ของชิปตระกูล arm64 เพื่อแสดงข้อความ `'Hello, World!'` ออกทางหน้าจอ

โปรแกรม 1.2: ภาษา Assembly ของชิปตระกูล ARM เพื่อแสดงข้อความ Hello, World! ออกทางหน้าจอ

```
.global _start

.section .data
message:
    .asciz "Hello, World!"

.section .text
_start:
    // การเรียกใช้ระบบบริการ write
    mov x0, 1           // ใส่ค่า 1 เพื่อระบุว่าจะพิมพ์ออกทางคอนโซล
```

```
    ldr x1, =message    // โหลดที่อยู่ของข้อความเข้า x1
    ldr x2, =13         // โหลดความยาวของข้อความ (13 ตัวอักษร) เข้า x2
    mov x8, 64          // ค่ารหัสการเรียกใช้บริการ write
    svc 0               // โหลดรหัสบริการเข้า x8 และเรียกใช้ระบบบริการ

// การเรียกใช้ระบบบริการ exit
    mov x0, 0           // ใส่ค่า 0 เพื่อระบุว่าจะออกจากโปรแกรมแล้ว
    mov x8, 93          // ค่ารหัสการเรียกใช้บริการ exit
    svc 0               // เรียกใช้ระบบบริการ exit
```

จะเห็นได้ว่าชุดคำสั่งที่ใช้ในภาษา Assembly นั้น มีรูปแบบการเขียนและทำความเข้าใจค่อนข้างยาก ต้องใช้เวลาในการเรียนรู้มาก แต่ข้อดีคือจะเป็นภาษาที่ทำงานได้เร็วที่สุดหากเขียนอย่างถูกวิธี นอกจากนี้การเขียนชุดคำสั่งในภาษา Assembly จะขึ้นอยู่กับว่าเขียนเพื่อใช้งานกับชิปหรือซีพียู (CPU) ใด ในสองตัวอย่างข้างต้นแสดงให้เห็นถึงความแตกต่างของชุดคำสั่งของภาษา Assembly ที่เขียนสำหรับซีพียูที่ใช้สถาปัตยกรรมแบบ x86 และ ARM เพื่อให้แสดงข้อความ `'Hello, World!'` ออกทางหน้าจอ

3. **ภาษาระดับสูง (High-level Language)** เป็นภาษาที่สร้างมาเพื่อให้ใกล้เคียงกับภาษาที่มนุษย์ใช้สื่อสารกันในชีวิตประจำวัน โดยคำสั่งต่าง ๆ มักใช้คำศัพท์ภาษาอังกฤษที่ง่ายต่อการจดจำและใช้งาน อาทิ `input` (ใช้สำหรับรับข้อมูล เช่น รับข้อมูลผ่านทางแป้นพิมพ์), `print` (สำหรับพิมพ์ข้อความที่ต้องการออกทางจอภาพและเครื่องพิมพ์), `read` (สำหรับอ่านข้อมูล เช่น จากไฟล์), `write` (สำหรับเขียนข้อมูลลงไฟล์), `if` (สำหรับตรวจสอบเงื่อนไข) ตัวอย่างของภาษาที่เป็นภาษาระดับสูง เช่น C, C++, Java, Python, Swift, Rust และ JavaScript

คราวนี้ลองมาดูตัวอย่างโปรแกรมภาษาไพทอนที่เป็นหนึ่งในภาษาระดับสูงเพื่อแสดงข้อความ `'Hello, World!'` ออกทางหน้าจอกันบ้าง ซึ่งใช้โค้ดเพียงหนึ่งบรรทัดเท่านั้น คือ

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

จะเห็นได้ว่าชุดคำสั่งของภาษาไพทอนนั้นใกล้เคียงกับภาษาที่มนุษย์ใช้ในการสื่อสารกัน เช่น ใช้ฟังก์ชันชื่อ `print()` เพื่อพิมพ์ข้อความที่ต้องการออกทางหน้าจอ ตัวโค้ดจะใช้เพียงแค่บรรทัดเดียวเท่านั้นเมื่อเทียบกับภาษา Assembly ซึ่งใช้เกือบ 20 บรรทัด อีกทั้งโค้ดดังกล่าวที่เขียนด้วยภาษาไพทอนสามารถทำไปทำงานได้กับเครื่องคอมพิวเตอร์ที่ใช้ระบบปฏิบัติการ และซีพียูที่มีสถาปัตยกรรมที่แตกต่างกันได้โดยไม่ต้องแก้ไข เช่น คำสั่งข้างต้นสามารถนำไปใช้ได้กับเครื่องที่ใช้ระบบปฏิบัติการ Windows, macOS และ Linux ได้

ลองมาดูตัวอย่างของภาษาระดับสูงที่ถูกนำไปใช้งานด้านต่าง ๆ

- **Python**: ใช้สำหรับการวิเคราะห์ข้อมูล, การเรียนรู้ด้วยเครื่อง (Machine Learning) และการเขียนสคริปต์ทั่วไปเพื่อทำงานแบบอัตโนมัติ รวมถึงการพัฒนาเว็บแอปพลิเคชัน
- **JavaScript**: ใช้สำหรับพัฒนาเว็บแอปพลิเคชัน

- **R**: เหมาะสำหรับการวิเคราะห์ข้อมูลและสถิติ
- **Swift**: ใช้สำหรับการพัฒนาแอปพลิเคชันบนแพลตฟอร์ม iOS, macOS, watchOS และ tvOS
- **PHP**: ใช้สำหรับการพัฒนาเว็บไซต์และแอปพลิเคชันที่ติดต่อกับฐานข้อมูล
- **C**: ใช้สำหรับการพัฒนาซอฟต์แวร์ที่ต้องการประสิทธิภาพสูง เช่น ระบบปฏิบัติการ (Windows, Linux, Android), Web server (Apache), โปรแกรมประยุกต์, โปรแกรมจัดการฐานข้อมูล (PostgreSQL, MySQL), เกมส์, โปรแกรมควบคุมฮาร์ดแวร์ (หุ่นยนต์)
- **C++**: ใช้สำหรับการพัฒนาซอฟต์แวร์ที่ต้องการประสิทธิภาพสูง เช่น โปรแกรมคอมพิวเตอร์ (Excel, Photoshop, Chrome browser), เกมส์, ระบบปฏิบัติการ (Windows), โปรแกรมควบคุมฮาร์ดแวร์ (หุ่นยนต์)
- **C#**: ใช้สำหรับการพัฒนาแอปพลิเคชันบนแพลตฟอร์ม Windows และเกมส์
- **Java**: ใช้สำหรับพัฒนาแอปพลิเคชันและเว็บ รวมถึงโปรแกรมบนเพื่อทำงานบนโทรศัพท์มือถือที่ใช้ระบบปฏิบัติการ Android, เกมส์ (Minecraft)
- **Go**: ใช้สำหรับพัฒนาซอฟต์แวร์ความเร็วสูงและน่าเชื่อถือ เช่น Docker, Kubernetes
- **Rust**: ใช้สำหรับการพัฒนาซอฟต์แวร์ที่ปลอดภัยและมีประสิทธิภาพสูง เหมาะสำหรับระบบที่ต้องการจัดการหน่วยความจำอย่างมีประสิทธิภาพ

จะเห็นได้ว่าภาษาคอมพิวเตอร์แต่ละภาษาก็จะมีจุดเด่นในแต่ละด้าน บางภาษาถูกออกแบบมาเพื่อให้ทำงานได้ดีในหลาย ๆ ด้าน (general-purpose language) เช่น C, Java, Python ส่วนบางภาษาอาจเน้นทำงานด้านใดด้านหนึ่งโดยเฉพาะ (domain-specific language) เช่น COBOL, SQL ดังนั้นเราจึงควรเลือกใช้ภาษาที่เหมาะสมกับงานหรือโครงการที่เราต้องการทำ เช่น หากต้องการเขียนเว็บแอปพลิเคชันก็ควรใช้ JavaScript หรือ PHP หากต้องการเขียนโปรแกรมสำหรับ iPhone ก็ควรใช้ Objective-C หรือ Swift

1.4 Interpreter กับ Compiler

ทั้ง Interpreter และ Compiler ล้วนเป็นเครื่องมือที่ใช้ในการแปลงภาษาโปรแกรมคอมพิวเตอร์ให้เป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจได้ ลองมาดูหลักการทำงานของ Interpreter และ Compiler ว่ามีจุดเด่นจุดด้อย อะไรบ้าง เหมาะสำหรับนำไปใช้งานด้านใด

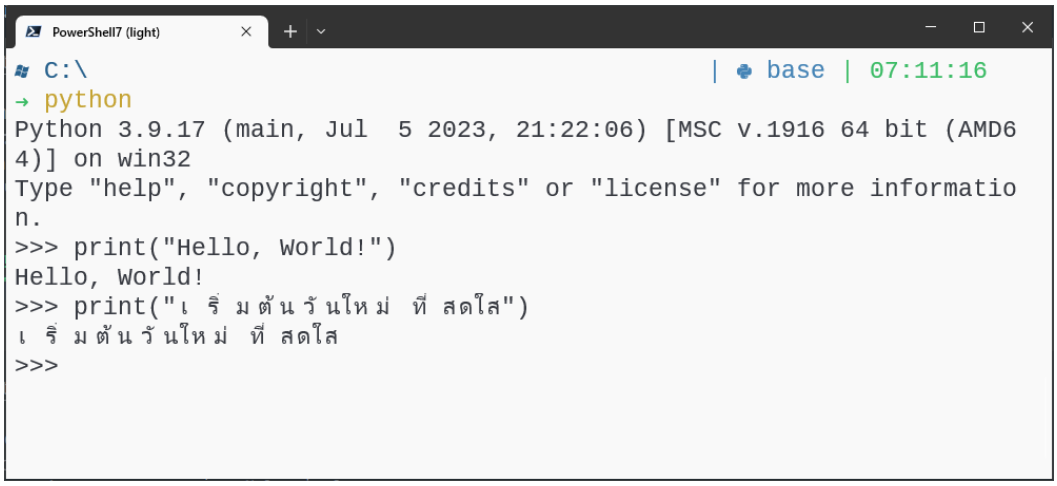
1.4.1 Interpreter

หากต้องการเข้าใจหลักการทำงานของ Interpreter ให้นึกถึงการทำงานของล่ามที่มีหน้าที่แปลประโยคคำพูดจากภาษาหนึ่งไปเป็นอีกภาษาหนึ่งแบบทันทีเพื่อให้ผู้รับสารเข้าใจในเนื้อหาที่ต้องการสื่อสารแบบประโยคต่อประโยคในทันที ในบริบทของคอมพิวเตอร์ ตัว Interpreter ก็คือโปรแกรมที่ดำเนินการแปลและประมวลผลโปรแกรมทีละคำสั่ง ซึ่งจะเทียบเท่ากับบทบาทของล่ามในการแปลภาษาคอมพิวเตอร์ให้

เป็นภาษาที่เครื่องคอมพิวเตอร์สามารถประมวลผลได้ทันที ตัวอย่างของภาษาที่ใช้หลักการทำงานแบบ Interpreter เช่น Python, JavaScript และ R

เพื่อให้เข้าใจการทำงานของ Interpreter ลองมาดูตัวอย่างการทำงานของ Python Interpreter

รูป 1.1: การทำงานของ Python Interpreter ที่รับและแปลทีละคำสั่งให้คอมพิวเตอร์ทำงาน



ขั้นตอนการทำงานของโปรแกรมเป็นดังนี้

- เมื่อมีการป้อนคำสั่ง `print("Hello, World!")` ตัว Python Interpreter ก็จะทำการแปลคำสั่งนี้ให้เป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจได้ แล้วแสดงผลลัพธ์ออกทางหน้าจอทันที
- จากนั้นเมื่อผู้ใช้ป้อนคำสั่งถัดมา `print("เริ่มต้นวันใหม่ที่สดใส")` ตัว Python Interpreter ก็จะทำการแปลคำสั่งนี้ แล้วแสดงผลลัพธ์ออกทางหน้าจอ

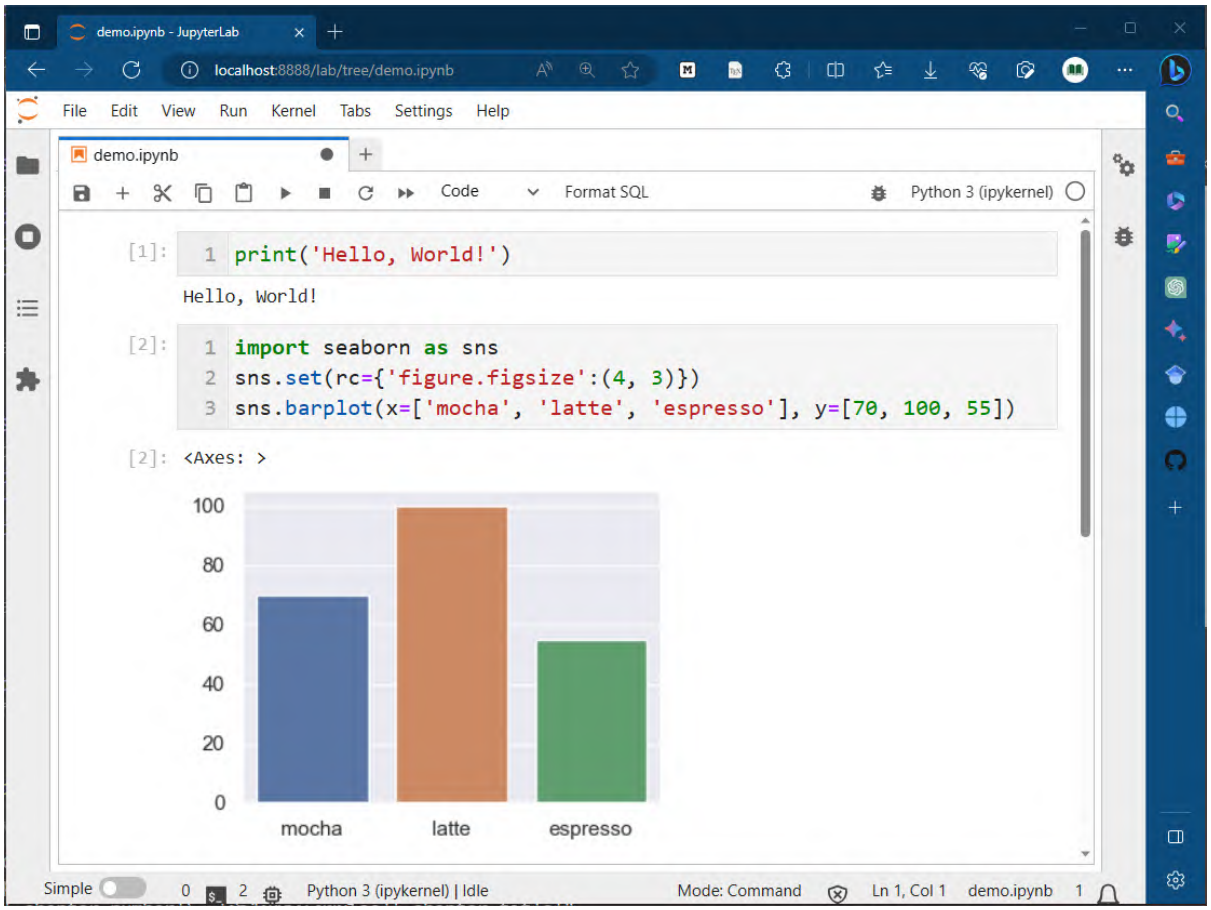
จะเห็นได้ว่าการทำงานของ Python Interpreter ในลักษณะนี้เปรียบเสมือนล่ามที่แปลชุดคำสั่งทีละคำสั่งให้เป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจเพื่อนำไปประมวลผลได้ทันทีนั่นเอง

> ⓘ prompt รอรับคำสั่งของ Python Interpreter
>
> `>>>` (chevron prompt) เป็นสัญลักษณ์แสดงความพร้อมในการรอรับคำสั่ง (prompt) ของตัว Python Interpreter โดยผู้ใช้สามารถพิมพ์คำสั่งของภาษาไพทอนเข้าไปหลังจากเห็น `>>>` นี้ปรากฏบนหน้าจอ

อีกรูปแบบหนึ่งของการใช้งาน Python ที่ได้รับความนิยมอย่างสูง คือ การเขียนโค้ดแล้วเก็บในไฟล์ .ipynb หรือ Jupyter Notebook ซึ่งเป็นรูปแบบของเอกสารที่ช่วยให้ผู้ใช้สามารถรวมโค้ดคอมพิวเตอร์ ข้อความ สมการ และภาพลงในเอกสารเดียว ทำให้ไฟล์รูปแบบนี้ได้รับความนิยมอย่างกว้างขวางในวงการวิทยาศาสตร์ข้อมูล การประมวลผล และวิเคราะห์ข้อมูล รวมถึงการสร้างโมเดลต่าง ๆ ที่ใช้ machine learning รวมถึงสาขาวิชาอื่น ๆ ที่ต้องการการวิเคราะห์ข้อมูลและการสร้างรายงาน

รูปถัดไปจะแสดงถึงการใช้ Interactive Python Notebook (`ipynb`) หรือเรียกสั้น ๆ ว่า Notebook เพื่อ run คำสั่งที่เขียนด้วยภาษาไพทอน

รูป 1.2: การใช้ Interactive Python Notebook (`ipynb`)



ในรูปข้างต้นจะมีชุดคำสั่งของภาษาไพทอนอยู่ในเซลล์ (cell) โดยผู้ใช้งานสามารถสั่งให้ Python Interpreter ทำการ run แต่ละเซลล์ได้ อย่างเช่น ในเซลล์แรกมีคำสั่ง `print('Hello, World!')` หาก run เซลล์นี้ก็จะได้ผลลัพธ์เป็น `Hello, World!` แสดงถัดจากเซลล์ที่มีชุดคำสั่ง ส่วนชุดคำสั่งที่อยู่ในเซลล์ถัดมา จะประกอบไปด้วย 3 คำสั่ง เพื่อใช้สำหรับแสดงกราฟข้อมูลยอดขายกาแฟแต่ละประเภท

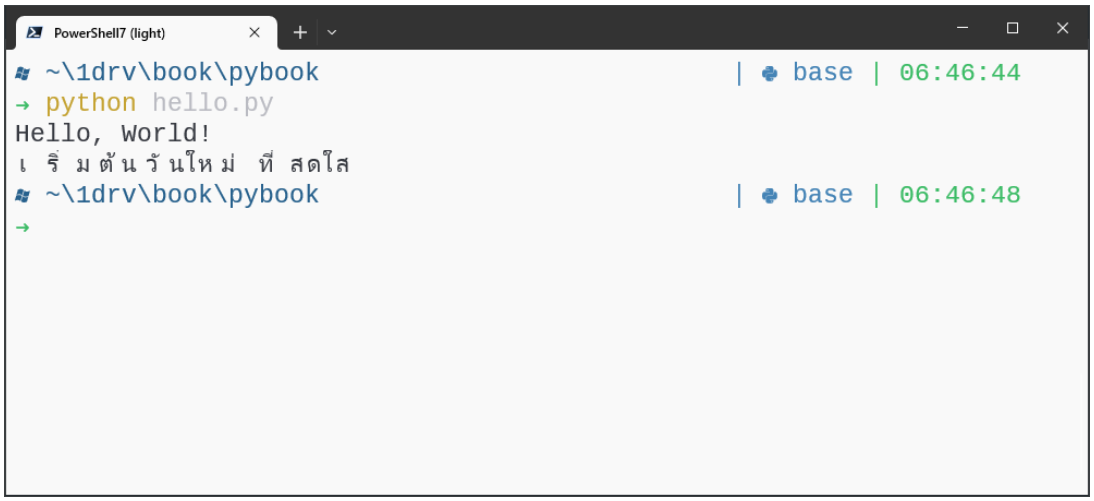
นอกจากการใช้งาน Python Interpreter แบบป้อนทีละคำสั่งแล้ว run คำสั่งนั้น ๆ ทันที ตัว Python Interpreter ยังสามารถ run คำสั่งที่ถูกเก็บไว้ในไฟล์ ซึ่งปกติจะมีนามสกุลเป็น `.py` เช่น ในตัวอย่างนี้จะสมมติว่ามีไฟล์ชื่อ `hello.py` ซึ่งภายในมีอยู่ 2 ชุดคำสั่ง เพื่อใช้แสดงข้อความที่เป็นภาษาอังกฤษและไทยออกทางหน้าจอ

โปรแกรม 1.3: โปรแกรมแรก Hello, World! (hello.py)

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

เราสามารถเรียกใช้ Python เพื่อให้คอมพิวเตอร์ปฏิบัติงานตามคำสั่งที่ถูกจัดเก็บอยู่ในไฟล์ได้ดังรูป

รูป 1.3: การเรียกใช้ Python เพื่อ run คำสั่งที่ถูกจัดเก็บอยู่ในไฟล์ hello.py



เมื่อใช้ Python Interpreter ทำการ run โปรแกรมข้างต้น ตัว Python Interpreter จะทำการแปลงคำสั่งทีละบรรทัดเริ่มจากบรรทัดแรก `print("Hello, World!")` แล้วทำการแสดงข้อความ `Hello, World!` บนหน้าจอ จากนั้นก็จะทำการแปลบรรทัดถัดไป `print("เริ่มต้นวันใหม่ที่สดใส")` แล้วจึงแสดงข้อความ เริ่มต้นวันใหม่ที่สดใส ออกมาทางหน้าจอ

ข้อดี

- การ run โปรแกรมแบบ real-time: โปรแกรมสามารถ run และแสดงผลลัพธ์ได้ทันทีเมื่อถูกแปล ซึ่งเป็นประโยชน์ในการพัฒนาและทดสอบโปรแกรมที่ต้องการตรวจสอบผลลัพธ์แบบ real-time เช่น การพัฒนาโมเดลที่ใช้ machine learning
- ความยืดหยุ่นในการพัฒนา: การใช้ Interpreter ช่วยลดเวลาในการพัฒนาโปรแกรมเนื่องจากไม่ต้องรอขั้นตอนการคอมไพล์ทุกส่วนของโปรแกรมก่อนที่จะเห็นผลลัพธ์ โดยเฉพาะหากโปรแกรมที่พัฒนานั้นมีขนาดใหญ่
- สะดวกในการแก้ไข: หากพบข้อผิดพลาดในโปรแกรมที่ถูก run ด้วย Interpreter สามารถแก้ไขและทดสอบโปรแกรมได้ทันทีโดยไม่ต้องคอมไพล์ใหม่ทั้งหมด

ข้อเสีย

- ประสิทธิภาพที่ต่ำกว่า Compiler: โดยทั่วไปแล้วโปรแกรมที่ถูกแปลและ run โดย Interpreter จะมีประสิทธิภาพโดยเฉพาะด้านความเร็วในการทำงานที่ต่ำกว่าโปรแกรมที่ถูกคอมไพล์และ run โดย Compiler
- ขึ้นอยู่กับ Interpreter: การ run โปรแกรมด้วย Interpreter จำเป็นต้องมีการติดตั้งตัว Interpreter สำหรับภาษาโปรแกรมที่ใช้ เช่น หากต้องการ run โปรแกรมภาษาไพทอนเครื่องคอมพิวเตอร์เครื่องนั้นจะต้องมี Python Interpreter เพื่อใช้ในการ run โปรแกรมที่เขียนด้วยภาษาไพทอน

- ต้องอาศัยซอร์สโค้ด (source code) ในการ run โปรแกรม: อาจไม่เหมาะกับโปรแกรมบางประเภทที่ผู้พัฒนาไม่ต้องการเปิดเผยซอร์สโค้ด

1.4.2 Compiler

คอมไพเลอร์ (Compiler) คือโปรแกรมที่ทำหน้าที่แปลงโปรแกรมจากภาษาคอมพิวเตอร์ที่มนุษย์เขียนให้เป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจได้ทั้งหมดก่อนที่จะ run โปรแกรมนั้น ๆ โดย Compiler จะทำการแปลและสร้างไฟล์ที่ถูกแปลงแล้ว (object file หรือ executable file) ที่สามารถ run ได้โดยตรงจากระบบปฏิบัติการ ตัวอย่างของภาษาที่ใช้แนวทางการแปลงโค้ดด้วย Compiler เช่น C, C++, Rust และ GO

โปรแกรมสำเร็จรูปส่วนใหญ่ที่เราใช้กันอยู่ในปัจจุบันมักเป็นโปรแกรมที่ถูกคอมไพล์มาแล้วสำหรับระบบปฏิบัติการนั้น ๆ เช่น Excel, Photoshop, Chrome browser ก็จะมีเวอร์ชันที่ถูกคอมไพล์เพื่อให้ทำงานได้กับเฉพาะระบบปฏิบัติการนั้น ๆ เช่น Windows, macOS, Linux

> ⓘ ไฟล์นามสกุล .exe บน Windows
>
> ไฟล์ที่มีนามสกุล .exe หรือ Executable File คือไฟล์ที่สามารถถูกเรียกใช้เป็นโปรแกรมหรือแอปพลิเคชันในระบบปฏิบัติการ Windows ของ Microsoft ไฟล์ประเภทนี้มักจะมีรหัสและข้อมูลที่จำเป็นสำหรับการประมวลผลและการทำงานของโปรแกรม และเมื่อผู้ใช้เปิดเรียกใช้ไฟล์ .exe ระบบปฏิบัติการจะเริ่มต้นการทำงานของชุดคำสั่งที่เก็บอยู่ในไฟล์นั้น
>
> ในระบบปฏิบัติการ Windows โปรแกรมที่ถูกคอมไพล์แล้วมักมีนามสกุลเป็น `.EXE` เช่น `EXCEL.EXE`, `notepad.exe`

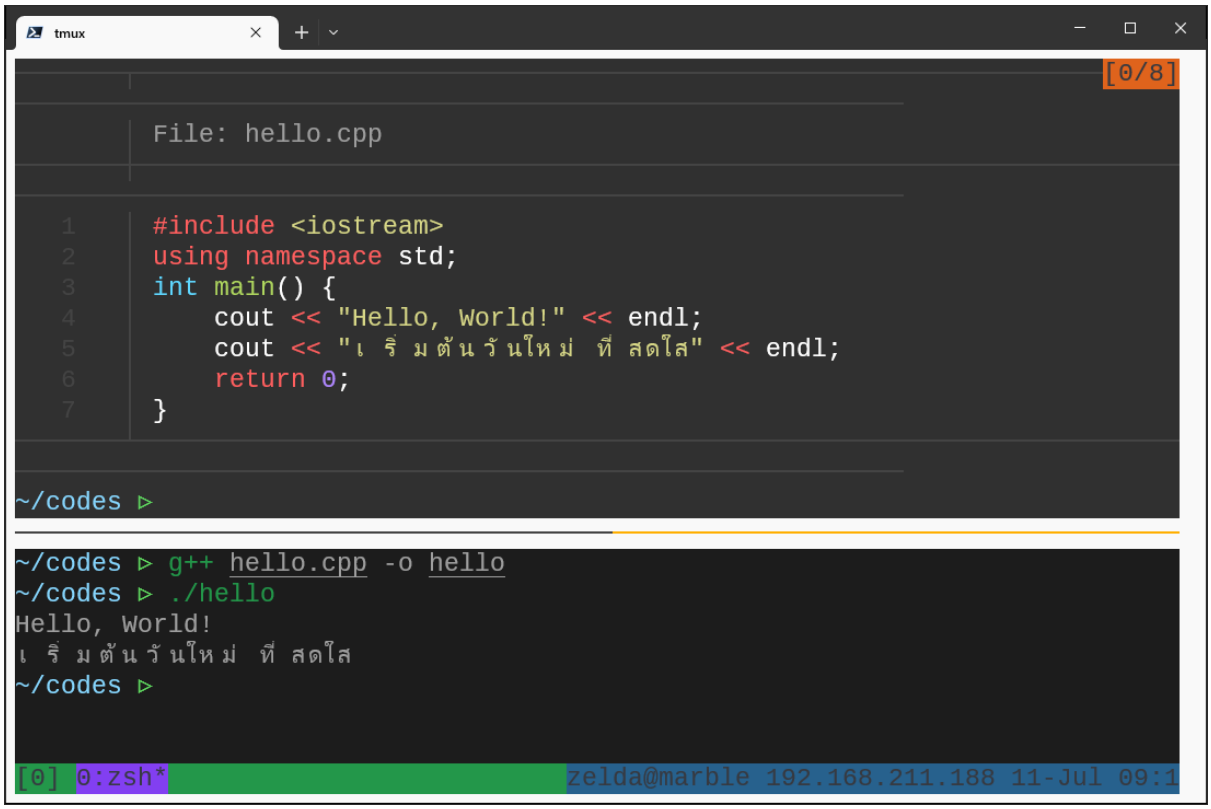
ตัวอย่างโค้ดภาษา C++ ที่ต้องผ่านการคอมไพล์ด้วย C++ Compiler ก่อนที่จะ run

โปรแกรม 1.4: ตัวอย่างโปรแกรมที่เขียนด้วยภาษา C++

```cpp
// ไฟล์ hello.cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!" << endl;
    cout << "เริ่มต้นวันใหม่ที่สดใส" << endl;
    return 0;
}
```

รูปถัดไปแสดงตัวอย่างการใช้ `g++` ที่เป็นหนึ่งใน C++ Compiler บนระบบปฏิบัติการ Linux ในการคอมไพล์โปรแกรม `hello.cpp` แล้วให้สร้างไฟล์ที่ถูกแปลงแล้ว (executable file) ชื่อ `hello` ถัดมาเป็นการ run โปรแกรม `hello` ซึ่งจะพิมพ์ผลลัพธ์ออกทางหน้าจอ

รูป 1.4: การใช้ g++ บนระบบปฏิบัติการ Linux ในการคอมไพล์ไฟล์ hello.cpp



ข้อดี

- ประสิทธิภาพสูง: โปรแกรมที่ถูกคอมไพล์โดย Compiler ทั้งหมดก่อนแล้วค่อย run จะมีประสิทธิภาพที่สูงกว่าโปรแกรมที่ถูกแปลและ run โดย Interpreter เนื่องจากโปรแกรมได้ถูกตรวจสอบความถูกต้องของไวยากรณ์ และประเภทของข้อมูลที่ใช้ว่าถูกต้องหรือไม่ตั้งแต่ขั้นตอนในการคอมไพล์ให้เป็นภาษาเครื่องตามสถาปัตยกรรมของซีพียูและระบบปฏิบัติการที่ต้องการเพื่อให้ได้เป็น executable file ก่อนที่จะนำไป run
- สามารถ run โปรแกรมได้ทันที: เมื่อคอมไพล์เสร็จสิ้นจะได้เป็น executable file ที่ผู้ใช้งานสามารถ run โปรแกรมได้ทันทีโดยไม่ต้องมี Compiler อยู่ในเครื่องที่ต้องการ run โปรแกรม เช่น การ run Excel, Photoshop, Chrome browser
- ไม่ต้องอาศัยซอร์สโค้ดในการ run: อาศัยเพียง executable file ที่ได้จากการคอมไพล์ไป run ได้เลย

ข้อเสีย

- เวลาที่ใช้ในการคอมไพล์: ในกระบวนการคอมไพล์ โปรแกรมจะถูกแปลงเป็นภาษาเครื่องทั้งหมดก่อนที่จะ run ซึ่งอาจใช้เวลาในการคอมไพล์ที่ยาวนานกว่าเมื่อใช้ Interpreter โดยเฉพาะหากโปรแกรมที่ต้องการคอมไพล์นั้นมีขนาดใหญ่มาก

- ความยุ่งยากในการแก้ไข: หากต้องการแก้ไขโปรแกรมที่คอมไพล์แล้ว จะต้องทำการแก้ไขที่ซอร์สโค้ดและคอมไพล์ใหม่ทุกครั้ง
- ต้องมีหลาย executable files ที่ถูกคอมไพล์เพื่อทำงานเฉพาะกับแต่ละระบบปฏิบัติการและสถาปัตยกรรมของซีพียูที่แตกต่างกัน

> คลิปศึกษาเพิ่มเติมบน YouTube
>
> Interpreter vs. Compiler

1.5 ตัวอย่างภาษาคอมพิวเตอร์

ลองมาดูตัวอย่างของการเขียนฟังก์ชันเพื่อแยกส่วนของชื่อและนามสกุลออกจากกัน พร้อมตัวอย่างการเรียกใช้งาน โดยใช้ภาษาต่าง ๆ กัน

Python

```python
# ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
def split_full_name(full_name):
    parts = full_name.split(" ", 1)
    first_name = parts[0]
    last_name = parts[1] if len(parts) > 1 else ""
    return first_name, last_name

full_name = "Peter Parker"
first_name, last_name = split_full_name(full_name)
print("First name:", first_name)
print("Last name:", last_name)
```

Javascript

```javascript
// ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
function splitFullName(fullName) {
    const parts = fullName.split(" ");
    const firstName = parts[0];
    const lastName = parts.length > 1 ? parts[1] : "";
    return [firstName, lastName];
}

const full_name = "Peter Parker";
const [first_name, last_name] = splitFullName(full_name);
```

```
console.log("First name:", first_name);
console.log("Last name:", last_name);
```

Swift

```
import Foundation

// ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
func splitFullName(_ full_name: String) -> (String, String) {
    let parts = full_name.components(separatedBy: " ")
    let first_name = parts[0]
    let last_name = parts.count > 1 ? parts[1] : ""
    return (first_name, last_name)
}

let full_name = "Peter Parker"
let (first_name, last_name) = splitFullName(full_name)
print("First name:", first_name)
print("Last name:", last_name)
```

R

```
# ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
splitFullName <- function(full_name) {
    parts <- strsplit(full_name, " ")[[1]]
    first_name <- parts[1]
    last_name <- ifelse(length(parts) > 1, parts[2], "")
    return(list(first_name, last_name))
}

full_name <- "Peter Parker"
name_parts <- splitFullName(full_name)
first_name <- name_parts[[1]]
last_name <- name_parts[[2]]

cat("First name:", first_name, "\n")
cat("Last name:", last_name, "\n")
```

C

```
#include <stdio.h>
#include <string.h>
```

```c
// ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
void splitFullName(char* full_name, char* first_name, char* last_name) {
    // ใช้ฟังก์ชัน strtok ในการแยกสตริงด้วยช่องว่างเป็นตัวแยก
    char* token = strtok(full_name, " ");
    if (token != NULL) {
        strcpy(first_name, token);
        token = strtok(NULL, " ");
        if (token != NULL) {
            strcpy(last_name, token);
        } else {
            strcpy(last_name, "");
        }
    } else {
        strcpy(first_name, full_name);
        strcpy(last_name, "");
    }
}

int main() {
    char full_name[] = "Peter Parker";
    char first_name[50];
    char last_name[50];

    splitFullName(full_name, first_name, last_name);

    printf("First name: %s\n", first_name);
    printf("Last name: %s\n", last_name);

    return 0;
}
```

C++

```cpp
#include <iostream>
#include <string>
#include <sstream>

// ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
void splitFullName(const std::string& full_name, std::string& first_name,
std::string& last_name) {
    std::istringstream iss(full_name);
```

```
    iss >> first_name >> last_name;

    if (iss.fail()) {
        last_name = "";
    }
}

int main() {
    std::string full_name = "Peter Parker";
    std::string first_name;
    std::string last_name;

    splitFullName(full_name, first_name, last_name);

    std::cout << "First name: " << first_name << std::endl;
    std::cout << "Last name: " << last_name << std::endl;

    return 0;
}
```

Java

```
public class SplitFullName {
    // ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
    public static void splitFullName(String full_name) {
        String[] parts = full_name.split(" ", 2);

        String first_name = parts[0];
        String last_name = parts.length > 1 ? parts[1] : "";

        System.out.println("First name: " + first_name);
        System.out.println("Last name: " + last_name);
    }

    public static void main(String[] args) {
        String full_name = "Peter Parker";
        splitFullName(full_name);
    }
}
```

Rust

```rust
// ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
fn split_full_name(full_name: &str) -> (&str, &str) {
    // ใช้การแบ่งสตริงด้วยช่องว่างเป็นตัวแยก
    let parts: Vec<&str> = full_name.split_whitespace().collect();

    // ตรวจสอบว่ามีสองส่วนหรือไม่
    if parts.len() == 2 {
        // หากมีสองส่วน คืนค่าชื่อและนามสกุล
    (parts[0], parts[1])
    } else {
        // หากไม่มีสองส่วน คืนค่าเป็นชื่อเต็มทั้งหมดในส่วนแรกและส่วนว่างในส่วนที่สอง
    (full_name, "")
    }
}

fn main() {
    let full_name = "Peter Parker";
    let (first_name, last_name) = split_full_name(full_name);
    println!("First name: {}", first_name);
    println!("Last name: {}", last_name);
}
```

Go

```go
package main

import (
    "fmt"
    "strings"
)

// ฟังก์ชันที่ใช้สำหรับแยกชื่อเต็มเป็นชื่อและนามสกุล
func splitFullName(fullName string) (string, string) {
    parts := strings.Fields(fullName)
    if len(parts) >= 2 {
        return parts[0], parts[1]
    }
    return fullName, ""
}
```

```
func main() {
    fullName := "Peter Parker"
    firstName, lastName := splitFullName(fullName)
    fmt.Println("First name:", firstName)
    fmt.Println("Last name:", lastName)
}
```

จากตัวอย่างข้างต้น ไม่ว่าจะเขียนด้วยภาษาใดก็ตาม ผลลัพธ์ที่ได้จะเป็นดังนี้

```
First name: Peter
Last name: Parker
```

จะเห็นได้ว่าแต่ละภาษาจะมีไวยากรณ์และรูปแบบคำสั่งที่แตกต่างกันออกไป แต่โดยส่วนใหญ่แล้วคำสั่งต่าง ๆ จะใช้คำศัพท์ภาษาอังกฤษที่สื่อความหมายในตัวเองที่เข้าใจได้ง่าย เช่น แต่ละภาษามักจะใช้คำสั่ง `print()` หรือใกล้เคียง เช่น `Println()` เพื่อแสดงข้อความออกทางหน้าจอ

ลองมาดูอีกสักตัวอย่างที่แสดงให้เห็นถึงการใช้ตัวดำเนินการทางคณิตศาสตร์ต่าง ๆ เพื่อนำมาสร้างฟังก์ชันสำหรับคำนวณโพรโมชันการรับประทานอาหารบุฟเฟต์ในรูปแบบ “มา x จ่าย y” เช่น มา 4 จ่าย 3 และวิธีการเรียกใช้งาน โดยสมมติว่า มีโพรโมชัน มา 4 จ่าย 3 บุฟเฟต์ปกติราคาหัวละ 100 บาท หากลูกค้ามา 5 คน จะต้องชำระเงินทั้งสิ้นเท่าไร

Python

```
def come_x_pay_y(pax, per_head, come_x=4, pay_y=3):
    total = (pax // come_x) * (pay_y * per_head) + (pax % come_x * per_head)
    return total

print(come_x_pay_y(5, 100, 4, 3)) # มา 5 คน ปกติบุฟเฟต์ราคาหัวละ 100 บาท
โดยมีโพรโมชัน มา 4 จ่าย 3 ดังนั้นยอดชำระจะอยู่ที่ 400 บาท
```

JavaScript

```
function come_x_pay_y(pax, per_head, come=4, pay=3) {
    let amt = Math.floor(pax / come) * (pay * per_head) + (pax % come) *
per_head;
    return amt;
}

console.log(come_x_pay_y(5, 100, 4, 3));
```

TypeScript

```
function come_x_pay_y(pax: number, per_head: number, come: number = 4, pay
: number = 3): number {
    const amt: number = (Math.floor(pax / come) * (pay * per_head)) +
((pax % come) * per_head);
    return amt;
}

console.log(come_x_pay_y(5, 100, 4, 3));
```

Swift

```
import Foundation

func come_x_pay_y(pax: Int, per_head: Int, come: Int = 4, pay: Int = 3) ->
Int {
    let amt = (pax / come) * (pay * per_head) + (pax % come) * per_head
    return amt
}

print(come_x_pay_y(pax: 5, per_head: 100, come: 4, pay: 3))
```

R

```
come_x_pay_y <- function(pax, per_head, come_x = 4, pay_y = 3){
    total <- (pax %/% come_x) * (pay_y * per_head) + (pax %% come_x *
per_head)
    return(total)
}

print(come_x_pay_y(5, 100, 4, 3))
```

C

```
#include <stdio.h>

int come_x_pay_y(int pax, int per_head, int come_x, int pay_y) {
    int total = (pax / come_x) * (pay_y * per_head) + (pax % come_x *
per_head);
    return total;
}
```

```c
int main() {
    printf("%d\n", come_x_pay_y(5, 100, 4, 3));
    return 0;
}
```

C++

```cpp
#include <cmath>
#include <iostream>
using namespace std;

int come_x_pay_y(int pax, int per_head, int come = 4, int pay = 3) {
    int amt = floor(pax / come) * (pay * per_head) + (pax % come) *
per_head;
    return amt;
}

int main() {
    cout << come_x_pay_y(5, 100, 4, 3) << endl;
}
```

Java

```java
public class come_x_pay_y {
    public static int come_x_pay_y(int pax, int per_head, int come_x, int
pay_y) {
        int total = (pax / come_x) * (pay_y * per_head) + (pax % come_x *
per_head);
    return total;
    }

    public static void main(String[] args) {
        System.out.println(come_x_pay_y(5, 100, 4, 3));
    }
}
```

Rust

```rust
fn come_x_pay_y(pax: i32, per_head: i32, come: i32, pay: i32) -> i32 {
    let amt = (pax / come) * (pay * per_head) + (pax % come) * per_head;
    return amt;
}
```

```rust
fn main() {
    println!("{}", come_x_pay_y(5, 100, 4, 3));
}
```

Go

```go
package main

import "fmt"

func come_x_pay_y(pax int, per_head int, come int, pay int) int {
    amt := (pax / come) * (pay * per_head) + (pax % come) * per_head
    return amt
}

func main() {
    fmt.Println(come_x_pay_y(5, 100, 4, 3))
}
```

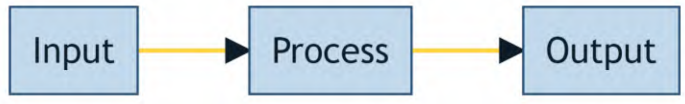
จากตัวอย่างข้างต้น ทางร้านมีโพรโมชัน มา 4 จ่าย 3 โดยบุฟเฟต์ราคาหัวละ 100 บาท ดังนั้นหากมาทาน 5 คน ยอดชำระจะอยู่ที่ 400 บาท ไม่ว่าจะเขียนด้วยภาษาใดก็ตาม

หากสังเกตให้ดีจะเห็นว่าไวยากรณ์หลาย ๆ อย่างของแต่ละภาษาข้างต้นล้วนใกล้เคียงกัน โดยเฉพาะในส่วนของการคำนวณทางคณิตศาสตร์ ซึ่งหากเรามีความรู้ในการเขียนโปรแกรมภาษาใดภาษาหนึ่งมาก่อนหน้า ก็สามารถอ่านทำความเข้าใจโค้ดที่เขียนด้วยภาษาอื่น รวมถึงการหัดเขียนได้ง่ายขึ้น เพราะหลักการในการออกแบบเขียนโปรแกรมไม่ว่าจะใช้ภาษาไหนก็ไม่ได้แตกต่างกัน ภาษาคอมพิวเตอร์เป็นเพียงเครื่องมือเท่านั้น แต่กระบวนการในการคิดแก้ปัญหา การแบ่งปัญหาออกเป็นส่วน ๆ ต่างหากที่เป็นสิ่งสำคัญในการพัฒนาโปรแกรมเพื่อสั่งงานคอมพิวเตอร์ให้ทำงานตามที่เราต้องการ

1.6 ขั้นตอนการทำงานของโปรแกรมคอมพิวเตอร์

โปรแกรมคอมพิวเตอร์มีวงจรการทำงานตามขั้นตอนต่อไปนี้

รูป 1.5: Flowchart แสดงวงจรการทำงานของโปรแกรมคอมพิวเตอร์



1. **นำเข้าข้อมูล (Input)**: โปรแกรมรับข้อมูลเข้ามาจากผู้ใช้หรือจากแหล่งข้อมูลอื่น ๆ เช่น แป้นพิมพ์ ไฟล์ หรือแม้กระทั่งเสียง

2. **ประมวลผล (Process)**: โปรแกรมดำเนินการตามคำสั่งหรืออัลกอริทึมที่กำหนด เพื่อดำเนินการประมวลผลข้อมูลตามที่ต้องการ เช่น การหาผลรวม การกรองข้อมูลตามเงื่อนไข
3. **ส่งผลลัพธ์ (Output)**: โปรแกรมส่งผลลัพธ์ออกมาให้ผู้ใช้หรือจัดเก็บไว้ในตัวแปรหรือไฟล์ เพื่อนำไปใช้ในการแสดงผลหรือนำไปประมวลผลต่อไป

มาดูตัวอย่างโปรแกรมที่เขียนด้วยภาษาไพทอนเพื่อแสดงถึงขั้นตอนการทำงานของร้านอาหารเมื่อลูกค้ามีการสั่งอาหาร

โปรแกรม 1.5: การทำงานของร้านอาหารเมื่อลูกค้ามีการสั่งอาหาร

```
def order_food():
    # 1. รับออร์เดอร์จากลูกค้า
    order = input("โปรดระบุอาหารที่ต้องการ: ")
    # 2. ส่งออร์เดอร์ไปยังครัว
    send_to_kitchen(order)
    # 3. รอการปรุงอาหารจากครัว
    cooking_status = wait_for_cooking()
    # 4. แสดงสถานะการปรุงอาหาร
    print("สถานะการปรุงอาหาร: ", cooking_status)
    # 5. รอรับอาหารที่พร้อมแล้ว
    food_ready = wait_for_food()
    # 6. นำอาหารมาเสิร์ฟให้ลูกค้า
    serve_food(food_ready)
    # 7. ลูกค้ารับอาหารและทาน
    print("ลูกค้ารับอาหารและทาน")

def send_to_kitchen(order):
    print("ส่งออร์เดอร์ไปยังครัว: ", order)

def wait_for_cooking():
    print("กำลังปรุงอาหาร...")
    return "กำลังปรุงอาหาร"

def wait_for_food():
    print("รอรับอาหารที่พร้อมแล้ว...")
    return "อาหารพร้อมแล้ว"

def serve_food(food):
    print("เสิร์ฟอาหาร: ", food)
```

```
# เรียกใช้งานฟังก์ชัน order_food()
order_food()
```

ในตัวอย่างนี้จะเป็นโครงของโปรแกรมแกรมเพื่อสาธิตให้เห็นถึงขึ้นตอนการทำงานในแต่ด้านที่เกิดขึ้นเมื่อลูกค้าสั่งอาหารจนถึงการเสิร์ฟอาหาร โดยฟังก์ชัน `order_food()` จะเป็นฟังก์ชันหลักที่แสดงขั้นตอนการทำงานในร้านอาหาร โดยเริ่มจากการรับออร์เดอร์จากลูกค้า แล้วส่งออร์เดอร์ไปยังครัวด้วยฟังก์ชัน `send_to_kitchen()` ถัดมาก็จะเรียกใช้ฟังก์ชัน `wait_for_cooking()` เพื่อรอการปรุงอาหาร เมื่ออาหารปรุงเสร็จ ก็จะแจ้งว่าอาหารปรุงเสร็จและพร้อมที่จะเสิร์ฟด้วยฟังก์ชัน `wait_for_food()` ขั้นตอนสุดท้ายจะเป็นการนำอาหารมาเสิร์ฟให้ลูกค้าด้วยฟังก์ชัน `serve_food()` โดยแต่ละขั้นตอนจะถูกแสดงผลลัพธ์บนหน้าจอ

1.7 โค้ดที่ดีเป็นอย่างไร

การเขียนโค้ดที่ดีนั้นเป็นทั้งศาสตร์และศิลป์ โดยหลักการแล้วโค้ดที่ดีต้องมีเราลักษณะดังต่อไปนี้

1. **ความชัดเจนและอ่านง่าย**: เริ่มจากการใช้ชื่อตัวแปรและฟังก์ชันที่เข้าใจง่าย มีการแยกส่วนของโค้ดออกเป็นฟังก์ชันหรือโมดูลที่มีหน้าที่แตกต่างกัน เพื่อให้โค้ดเป็นระเบียบและง่ายต่อการแก้ไขหรือปรับปรุงในอนาคต
2. **มีเอกสารและคำอธิบายโค้ด**: ควรมีการใส่คำอธิบายการทำงานของโค้ดหรือมักเรียกว่าคอมเมนต์ (comment) เพื่ออธิบายแนวคิดหรือวิธีการทำงานของส่วนของโค้ด ซึ่งจะช่วยผู้เขียน รวมถึงบุคคลอื่นที่อ่านหรือต้องการแก้ไขโค้ดในอนาคตเข้าใจโครงสร้างและวิธีการทำงานได้ดียิ่งขึ้น
3. **มีการทดสอบ**: ในการเขียนโค้ดที่ดี ควรมีการทดสอบโค้ดเพื่อตรวจสอบว่ามันทำงานตามที่คาดหวัง
4. **ปฏิบัติตามหลักการและมาตรฐานที่ได้รับการยอมรับ**: เริ่มตั้งแต่วิธีการตั้งชื่อตัวแปร (เช่น การใช้ snake_case) ชื่อฟังก์ชัน ชื่อคลาส (ใช้ Pascal Case) รวมถึงการเยื้อง (indentation เช่น ใช้ 2 หรือ 4 ช่องว่าง) ให้เป็นไปตามมาตรฐานที่ทีมงานกำหนด
5. **จัดกลุ่มโค้ดที่เกี่ยวข้องกันเข้าด้วยกัน**: ใช้โครงสร้างข้อมูลและอัลกอริทึมที่เหมาะสมในการแก้ปัญหา และใช้ฟังก์ชันหรือคลาสให้เหมาะสมกับหน้าที่ที่ต้องการ เช่น คลาสสำหรับการบีบอัดข้อมูล คลาสสำหรับการสุ่มตัวเลข คลาสสำหรับทำงานกับรูปภาพ
6. **โค้ดที่ยืดหยุ่น:** ควรเขียนโค้ดที่สามารถปรับปรุงและขยายขอบเขตได้ง่าย ใช้ตัวแปรและค่าคงที่ที่เหมาะสมเพื่อให้สามารถเปลี่ยนแปลงค่าได้โดยไม่ต้องแก้ไขโค้ดทั้งหมด
7. **ลดการซ้ำซ้อน**: โค้ดควรถูกออกแบบให้สามารถลดการซ้ำซ้อนของโค้ดและการสร้างโมดูลที่นำกลับมาใช้ซ้ำได้

8. **มีการจัดการข้อผิดพลาด**: ในการเขียนโค้ดที่ดี เราควรคำนึงถึงการจัดการข้อผิดพลาด โดยใช้รูปแบบการดักจับและการจัดการข้อผิดพลาดที่เหมาะสม เพื่อให้โปรแกรมสามารถทำงานได้อย่างเป็นระเบียบและปลอดภัย
9. **เน้นเรื่องความปลอดภัย**: โค้ดที่ดีควรให้ความสำคัญกับความปลอดภัย โปรแกรมควรมีการจัดการข้อมูลที่ปลอดภัยและการรักษาความลับ

1.8 สรุป

โปรแกรมคอมพิวเตอร์เป็นชุดของคำสั่งที่ถูกออกแบบมาเพื่อสั่งการและควบคุมเครื่องคอมพิวเตอร์ให้ทำงานตามที่ผู้พัฒนาต้องการ โดยชุดคำสั่งเหล่านี้จะถูกเขียนด้วยภาษาโปรแกรมที่เครื่องคอมพิวเตอร์สามารถแปลและประมวลผลได้

ความรู้เกี่ยวกับการเขียนโปรแกรมคอมพิวเตอร์ไม่เพียงแต่เป็นทักษะพื้นฐานที่จำเป็นสำหรับการเข้าใจและการสร้างเทคโนโลยีดิจิทัล แต่ยังเป็นส่วนสำคัญในการทำงานของระบบดิจิทัลที่เราใช้งานในชีวิตประจำวัน

Bill Gates หนึ่งในผู้ก่อตั้งบริษัท Microsoft ให้คำแนะนำไว้ว่าหากต้องการเป็นนักพัฒนาโปรแกรมที่ดี ควรต้องมีเราสมบัติดังต่อไปนี้

1. Don’t Overthink, Just Dive In (อย่าคิดมากเกินไป ขอแค่เริ่มลงมือทำ)
2. Know Your Tools Well — Really Well (รู้จักเครื่องมือของเราเป็นอย่างดี - ดีสุด ๆ)
3. Get Good at Reading the Code (หัดอ่านโค้ดให้เก่ง)
4. Learn To Make Things as Simple as Possible (เรียนรู้ที่จะทำให้สิ่งต่าง ๆ ง่ายที่สุดเท่าที่จะเป็นไปได้)
5. Learn to Work In The Group (เรียนรู้ที่จะทำงานเป็นทีม)
6. Visualize First Then Create It (เห็นภาพก่อนว่าต้องการอะไรแน่แล้วจึงค่อยลงมือสร้างมัน)
7. Know the Joy of Creating Something (รู้ถึงความสุขในการสร้างสรรค์สิ่งนั้น ๆ ขึ้นมา)

อ้างอิงจาก: Here’s Bill Gates’s advice to new programmers. It should not be ignored. | by Somnath Singh. https://javascript.plainenglish.io/heres-bill-gates-s-advice-to-new-programmers-it-should-not-be-ignored-33e31378f0ae.

แบบฝึกหัด

1. ทำไมเราถึงควรที่จะเรียนการเขียนโปรแกรมคอมพิวเตอร์ พร้อมยกตัวอย่าง
2. อธิบายขั้นตอนการทำงานของโปรแกรมคอมพิวเตอร์โดยสังเขป
3. อธิบายความแตกต่างระหว่าง Interpreter และ Compiler

2. รู้จักภาษาไพทอน Python

วัตถุประสงค์การเรียนรู้

- รู้จักภาษาไพทอน
- การใช้ `indentation` ในการเขียนโค้ด
- การใช้ฟังก์ชัน `print()` แบบต่าง ๆ
- การใส่คำอธิบายการทำงานของโปรแกรม (`comment`) ลงในโค้ด

2.1 ภาษาไพทอน

ภาษาไพทอนเป็นภาษาคอมพิวเตอร์ที่ถูกสร้างขึ้นโดย Guido van Rossum ในช่วงปลายทศวรรษ 1980 ขณะที่เขาทำงานที่สถาบันวิจัย Centrum Wiskunde & Informatica (CWI) ในประเทศเนเธอร์แลนด์ และได้เปิดตัวต่อสาธารณชนครั้งแรกในปี 1991 ในปัจจุบัน (ปี 2023) ภาษาไพทอนได้พัฒนามาถึงเวอร์ชัน 3.12 ซึ่งทำงานได้บนระบบปฏิบัติการต่าง ๆ ไม่ว่าจะเป็น Windows, macOS และ Linux

ภาษาไพทอนถูกออกแบบให้เป็นภาษาที่อ่านและเรียนรู้เข้าใจได้ง่าย เป็นโอเพนซอร์ส (open source) มีชุมชนนักพัฒนาจำนวนมากช่วยในการพัฒนาตัวความสามารถของภาษา และโมดูลการทำงานต่าง ๆ อย่างต่อเนื่อง ตัวภาษาเองมีความสามารถสูง เป็นภาษาแบบเอนกประสงค์ และมีโมดูลสำหรับงานด้านต่าง ๆ อยู่มากมาย ทำให้สามารถนำไปประยุกต์ใช้กับงานได้หลากหลาย เช่น ทำการวิเคราะห์ข้อมูล, AI, Machine learning, Web application ฯลฯ ทำให้ได้รับความนิยมอย่างสูงมากในปัจจุบัน

จากผลสำรวจของ Stack Overflow ในปี 2023 ภาษาไพทอนได้รับความนิยมสูงสุดเป็นอันดับที่ 3 เป็นรองแค่เพียง JavaScript และ HTML/CSS เท่านั้น

> ⓘ **`open source` คืออะไร**
>
> โอเพนซอร์ส (open source) เป็นประเภทของการพัฒนาซอฟต์แวร์ที่มีการเผยแพร่ซอร์สโค้ดภายใต้ใบอนุญาตที่ผู้ถือลิขสิทธิ์ให้สิทธิ์แก่ผู้ใช้ในการใช้ศึกษาเปลี่ยนแปลงและแจกจ่ายซอฟต์แวร์และซอร์สโค้ดให้กับทุกคน โดยซอฟต์แวร์โอเพ่นซอร์สอาจได้รับการพัฒนาในลักษณะสาธารณะร่วมกัน

ลองมาดูตัวอย่างของบริษัทและองค์กรชั้นนำต่าง ๆ ในโลกที่นำเอาภาษาไพทอนไปใช้งาน อาทิ Microsoft, Apple, Google, Facebook, Netflix, Spotify, Uber, Reddit, Airbnb, NASA และ Sony

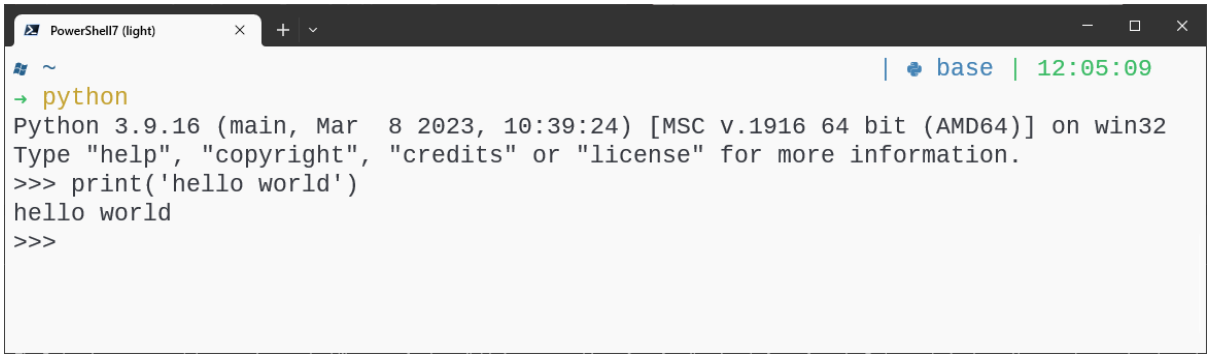
> **การติดตั้ง Python บน Windows**
>
> การติดตั้ง Python ทำได้หลายวิธี โดยส่วนตัวของผู้เขียนชอบวิธีการติดตั้งผ่าน `miniconda` โดยดาวน์โหลดได้ที่ https://docs.conda.io/en/main/miniconda.html
>
> การติดตั้ง Python บน Windows

2.1.1 โปรแกรมแรก

หลังจากที่ทำการติดตั้ง Python แล้ว เราสามารถเรียกใช้งาน Python โดยใช้ Command Prompt (หรือ PowerShell) หรือโปรแกรมแก้ไขข้อความที่สนับสนุน Python เช่น IDLE, Notepad, Visual Studio Code เพื่อเขียนและ run โปรแกรมที่เขียนด้วยภาษาไพทอนได้

ตัวอย่างการเรียกใช้งาน Python บน Windows พร้อมป้อนโค้ดสำหรับแสดงข้อความ `hello world` ออกทางหน้าจอ

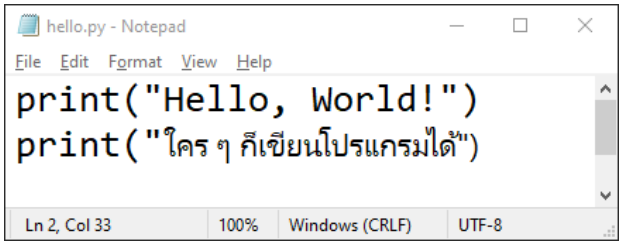
รูป 2.1: การเรียกใช้งาน Python บน Windows



> **>>> คืออะไร**
>
> สัญลักษณ์ `>>>` เป็น prompt (ตัวรอรับคำสั่ง) ที่แสดงถึงความพร้อมของ Python Interpreter ที่รอให้ผู้ใช้ป้อนชุดคำสั่งเข้าไป

หากต้องการเขียนโค้ดที่มีหลายบรรทัด ขอแนะนำให้ใช้ text editor เช่น Notepad หรือ IDE เช่น Visual Studio Code, sublime

รูป 2.2: การเขียนโค้ดด้วย `notepad.exe` บนระบบปฏิบัติการ Windows



> ⓘ IDE คืออะไร
>
> **IDE** (Integrated Development Environment) คือโปรแกรมที่มีความสามารถในการสนับสนุนและช่วยในกระบวนการพัฒนาซอฟต์แวร์หรือแอปพลิเคชัน โดยเฉพาะอย่างยิ่งในการเขียนและแก้ไขโค้ด มักมีคุณสมบัติที่อำนวยความสะดวกในการเขียนโค้ด เช่น code completion การจัดรูปแบบโค้ด (code formatting) การตรวจจับข้อผิดพลาด (error detection) การทำ version control และอื่น ๆ
>
> **Visual Studio Code** หรือเรียกย่อ ๆ ว่า VS Code เป็นตัว IDE ที่นิยมใช้กันอย่างแพร่หลายในการเขียนโค้ดของ Python เป็น IDE ที่ให้ติดตั้งและใช้งานได้ฟรี ทำงานได้กับระบบปฏิบัติการหลัก ๆ ไม่ว่าจะเป็น Windows, macOS และ Linux รวมถึงมีชุมชนที่ช่วยเขียนส่วนต่อขยายความสามารถ (extensions) ให้กับ VS Code เป็นจำนวนมาก ทำให้การเขียนโค้ดโดยใช้ VS Code เป็นไปอย่างมีประสิทธิภาพและรวดเร็วยิ่งขึ้น
>
> ดาวน์โหลดและติดตั้ง Visual Studio Code ได้ที่ https://code.visualstudio.com/download
>
> 📺 การติดตั้ง Visual Studio Code บน Windows

ตัวอย่างของโค้ดแบบเบื้องต้น พร้อมผลลัพธ์ที่ได้จากการทำงาน

```
print("Hello, World!")
print("ใคร ๆ ก็เขียนโปรแกรมได้")
```

ผลลัพธ์:

```
Hello, World!
ใคร ๆ ก็เขียนโปรแกรมได้
```

ตัวอย่างโค้ดสำหรับการคำนวณพื้นที่สี่เหลี่ยมผืนผ้าที่มีความกว้างเท่ากับ 20 เมตร และมีความยาวเท่ากับ 5 เมตร

```
print("พื้นที่ = ", 20 * 5, "ตารางเมตร")
```

ผลลัพธ์:

```
พื้นที่ = 100.0 ตารางเมตร
```

เราสามารถเก็บค่าความกว้างและความยาวไว้ในตัวแปรก่อนแล้วค่อยนำมาคำนวณ เช่น ใช้ตัวแปร `w` และ `h` เพื่อทำการเก็บค่าความกว้างและยาว แล้วนำมาคำนวณหาพื้นที่ภายหลัง

```
w = 20
h = 5
print("พื้นที่ = ", w * h, "ตารางเมตร")
```

ผลลัพธ์:

```
พื้นที่ = 100.0 ตารางเมตร
```

ตัวอย่างถัดไปจะเป็นโปรแกรมที่มีการรับค่าความกว้างและยาวผ่านทางแป้นพิมพ์จากผู้ใช้งาน แล้วนำมาคำนวณหาพื้นที่พร้อมแสดงผลทางหน้าจอ

```
w = float(input("กว้าง: "))
h = float(input("ยาว: "))
print("พื้นที่ = ", w * h, "ตารางเมตร")
```

ผลลัพธ์:

```
กว้าง: 20
ยาว: 5
พื้นที่ = 100.0 ตารางเมตร
```

2.1.2 ทำไมภาษาไพทอนถึงได้รับความนิยมอย่างแพร่หลาย

ภาษาไพทอนได้รับความนิยมในหลากหลายแวดวงด้วยเหตุผลหลายประการดังต่อไปนี้

1. **อ่านและเข้าใจง่าย**: ไพทอนเป็นภาษาโปรแกรมที่อ่านและเข้าใจได้ง่าย เนื่องจากมีโครงสร้างที่กระชับ อีกทั้งคำสั่งต่าง ๆ ก็มาจากคำพื้นฐานที่มนุษย์ใช้สื่อสารกันในชีวิตประจำวัน เช่น หากต้องการให้พิมพ์ข้อความออกทางหน้าจอก็ใช้ `print('your text')` หากต้องการรับค่าผ่านทางแป้นพิมพ์ก็ใช้ฟังก์ชัน `input()` ซึ่งทำให้ผู้เรียนและนักพัฒนาสามารถเข้ามาเรียนรู้และเขียนโปรแกรมได้ง่ายและเร็วขึ้น
2. **มีความหลากหลายของไลบรารี**: ไพทอนมีชุดไลบรารีและเครื่องมือมากมายที่สนับสนุนการพัฒนาโปรแกรมหลากหลายรูปแบบ ไม่ว่าจะเป็นการจัดการข้อมูลทางธุรกิจ, การวิเคราะห์ข้อมูล, การเชื่อมต่อฐานข้อมูล, การพัฒนาเว็บไซต์, การทำ machine learning และอื่น ๆ การมีชุดไลบรารีที่หลากหลาย ช่วยให้ผู้พัฒนาสามารถใช้งานและแก้ไขปัญหาต่าง ๆ ได้อย่างรวดเร็ว
3. **การสนับสนุนและชุมชนที่แข็งแกร่ง**: ไพทอนมีชุมชนผู้ใช้ที่ใหญ่และแข็งแกร่ง มีการสนับสนุนและแบ่งปันความรู้ รวมถึงมีการพัฒนาและอัปเดตภาษาอย่างต่อเนื่อง ซึ่งช่วยให้ผู้เรียนและนักพัฒนาสามารถเรียนรู้และแก้ไขปัญหาได้อย่างรวดเร็ว และมีทรัพยากรการเรียนรู้ออนไลน์ที่มีมากมาย เช่น คู่มือการใช้งาน, คอร์สออนไลน์ และฟอรัมการสนทนา

2.2 รูปแบบของภาษาไพทอน

โดยปกติเราจะเก็บโค้ดเอาไว้ในไฟล์ที่มีนามสกุล `.py` เช่น `promotion.py` การเก็บโค้ดไว้ในไฟล์ช่วยให้เราสามารถเก็บรวบรวมโค้ดเพื่อนำไปใช้ซ้ำและแชร์กับผู้อื่นได้อย่างสะดวกยิ่งขึ้น นอกจากนี้ยังช่วยให้การ

พัฒนาโปรแกรมง่ายขึ้น เนื่องจากเราสามารถแยกโค้ดเป็นไฟล์ย่อย ๆ ได้ตามความเหมาะสม เพื่อให้โค้ดมีความชัดเจนและง่ายต่อการบริหารจัดการ รวมถึงการแก้ไขเพิ่มเติมความสามารถในการทำงานในภายหน้า

> ⚠ Python เป็นภาษาแบบ case sensitive
>
> ภาษาไพทอนเป็นภาษาแบบ case sensitive เหมือนกับภาษา C, C++, JavaScript นั่นคือ ตัวอักษรพิมพ์ใหญ่และพิมพ์เล็กมีความแตกต่างกัน เช่น
>
> ```
> age = 20
> Age = 18
> AGE = 25
> ```
>
> ในที่นี้ตัวแปร `age`, `Age` และ `AGE` จะเป็นคนละตัวกันในภาษาไพทอน
>
> ```
> def calc_distance():
>     pass
>
> def Calc_Distance():
>     pass
> ```
>
> สำหรับชื่อฟังก์ชัน `calc_distance()` และ `Calc_Distance()` ก็จะถือเป็นคนละฟังก์ชันเช่นกัน

ในภาษาไพทอนการทำ `indentation` (การเยื้องส่วนของโค้ดด้วยการเว้นวรรคด้วยช่องว่าง (space) หรือแท็บด์ tab) เป็นส่วนสำคัญของโค้ด เพราะภาษาไพทอนใช้การเยื้องเพื่อกำหนดขอบเขตของโค้ดบล็อก (code block) เช่น การสร้างฟังก์ชัน (function), การสร้างลูป (loop) และการเขียนเงื่อนไข (condition)

ในภาษาไพทอนมักนิยมใช้ช่องว่าง (space) 4 ช่อง หรือจะใช้ tab ในการจัดกลุ่มของโค้ดบล็อก ซึ่งจำนวนช่องว่างหรือจำนวน tab ที่ใช้เว้นวรรคสำหรับโค้ดบล็อกเดียวกันจะต้องเท่ากัน

ลองมาดูตัวอย่างของการใช้การเยื้องในโค้ดภาษาไพทอนเพื่อจัดโค้ดบล็อกที่เกี่ยวข้องกันไว้ด้วยกัน

โปรแกรม 2.1: การใช้ `indentation` เพื่อกำหนดขอบเขตของโค้ดบล็อก

```
def rectangle_area(w, h):
    area = w * h              # 1
    return area               # 1

def citizen(age):
    if age > 60:              # 2
        print("senior")       # 2
        print("50% discount") # 2
```

```
    else:                       # 2
        print("young")          # 2

your_age = 28                   # 3
citizen(your_age)               # 3
```

โค้ดจุดที่ 1 จะเป็นโค้ดการทำงานของฟังก์ชันชื่อ `rectangle_area()` ซึ่งมีการเว้นวรรค 4 ช่อง เพื่อให้รู้ว่าโค้ด 2 บรรทัดนี้เป็นโค้ดบล็อกที่อยู่ภายใต้ฟังก์ชัน `rectangle(area)` ส่วนโค้ดในจุดที่ 2 จะเป็นโค้ดบล็อกของฟังก์ชัน `citizen()` โดยที่ภายในฟังก์ชันนี้ ก็ยังประกอบด้วยโค้ดบล็อกย่อยสำหรับคำสั่ง `if age > 60:` เป็นจริง และโค้ดบล็อกภายใต้ `else` ในกรณีไม่เป็นจริง

ส่วนโค้ดในจุดที่ 3 เริ่มต้นที่คอลัมน์แรก ไม่ได้มีการเว้นวรรคให้อยู่ภายใต้โค้ดบล็อกใด ๆ จะเป็นโค้ดที่ถูกทำงานก่อน โดยในโค้ดเริ่มจากการกำหนดค่าให้ตัวแปร `your_age` เท่ากับ `28` ถัดมาจะมีการเรียกใช้งานฟังก์ชัน `citizen()` โดยผ่านค่า `your_age` เข้าไปยังพารามิเตอร์ `age` ของฟังก์ชัน `citizen()`

2.3 ฟังก์ชัน print()

ฟังก์ชัน `print()` ใช้สำหรับแสดงผลบนหน้าจอ ถือเป็นคำสั่งที่มีการใช้มากที่สุดคำสั่งหนึ่ง มีรูปแบบการใช้งานหลัก ๆ ดังนี้

รูปแบบ	คำอธิบาย	ตัวอย่างการใช้งาน
`print(value)`	พิมพ์ค่า `value` และขึ้นบรรทัดใหม่	`print("Hello, World!")`
`print(value1, value2, ..., sep=' ', end='\n')`	พิมพ์ค่า `value1`, `value2`, … โดยใช้ตัวคั่น `sep` และสิ้นสุดด้วย `end`	`print("Hello", "World", sep=", ", end="!")`
`print(value1, value2, ..., end='')`	พิมพ์ค่า `value1`, `value2`, … โดยไม่ขึ้นบรรทัดใหม่	`print("Hello", "World", end="")`

โปรแกรม 2.2: การใช้คำสั่ง `print()` เบื้องต้น

```
print("Hello, World!")
print("mocha", "latte", "espresso")
print("one", "two", "three", sep=">")
print("area = ", 5 * 10, "sq.m.")
print("coconut", "mango", sep=" ", end="")
print("POMELO")
print("watermelon")
```

ผลลัพธ์:

```
Hello, World!
mocha latte espresso
one>two>three
area =  50 sq.m.
coconut mangoPOMELO
watermelon
```

2.4 คำอธิบายโค้ด (comment)

การเขียนคำอธิบายเพิ่มเติมในโค้ดหรือคอมเมนต์ (comment) เป็นการอธิบายแนวคิดหรือวิธีการทำงานของโค้ด ช่วยให้ผู้พัฒนารวมถึงบุคคลอื่นที่อ่านหรือต้องการแก้ไขโค้ดในอนาคตเข้าใจโครงสร้างและวิธีการทำงานได้ดียิ่งขึ้น นอกจากนี้เรายังใช้คอมเมนต์เพื่อเก็บบันทึกเกี่ยวกับการแก้ไขหรือปรับปรุงโค้ด ทำให้สามารถติดตามและเข้าใจการเปลี่ยนแปลงในอนาคตได้

ตัว Python Interpreter จะไม่ประมวลผลคำสั่งใด ๆ ที่อยู่ในคอมเมนต์ ดังนั้นคำสั่งที่เป็นคอมเมนต์จะไม่มีผลต่อการทำงานของโค้ด

ในภาษาไพทอน เราสามารถเขียนคอมเมนต์ได้ 3 วิธี

1. คอมเมนต์แบบบรรทัดเดียว ทำได้โดยใช้เครื่องหมาย # นำหน้าคำอธิบายโค้ด เช่น

```
# นี่คือคอมเมนต์บรรทัดเดียว
# another single line comment

# อัตราภาษีมูลค่าเพิ่ม 7%
vat = .07

# ราคาสินค้าก่อนรวมภาษีมูลค่าเพิ่ม
price_before_vat = 100
```

2. คอมเมนต์ต่อท้ายบรรทัดของโค้ด เราสามารถเขียนคอมเมนต์ต่อท้ายบรรทัดของโค้ดได้ โดยใช้เครื่องหมาย # หลังจากโค้ด เช่น

```
vat = .07 # อัตราภาษีมูลค่าเพิ่ม 7%
price_before_vat = 100 # ราคาสินค้าก่อนรวมภาษีมูลค่าเพิ่ม
```

3. คอมเมนต์แบบหลายบรรทัด เราสามารถเขียนคอมเมนต์หลายบรรทัดโดยใช้เครื่องหมาย ''' (triple single quote) หรือ """ (triple double quote) คลุมข้อความที่เป็นคอมเมนต์ เช่น

```
'''
นี่คือคอมเมนต์หลายบรรทัด ด้วย triple single quote
```

```
บรรทัดที่ 2
บรรทัดที่ 3
'''
c = 27
f = 1.8 * c + 32
print(f)
```

หรือ

```
"""
นี่คือคอมเมนต์หลายบรรทัด ด้วย triple double quote
บรรทัดที่ 2
บรรทัดที่ 3
"""
c = 27
f = 1.8 * c + 32
print(f)
```

2.5 คำสงวน (reserved keywords) ของภาษาไพทอน

คำสงวน (`reserved keywords`) คือคำที่มีการกำหนดให้มีหน้าที่และความหมายเฉพาะเจาะจงล่วงหน้าแล้วในภาษาคอมพิวเตอร์ จึงไม่สามารถนำมาใช้เป็นชื่อตัวแปรหรือชื่อฟังก์ชันในได้

ตาราง 2.1: คำสงวน (reserved keywords) ในภาษาไพทอน

คำสงวน	คำอธิบาย
`False`	ค่าบูลีน**เท็จ**
`True`	ค่าบูลีน**จริง**
`None`	ค่าว่าง
`and`	ตัวดำเนินการ**และ**
`or`	ตัวดำเนินการ**หรือ**
`not`	ตัวดำเนินการ**ไม่**
`in`	ตรวจสอบค่าใน `list` หรือตัวแปร
`is`	ตรวจสอบเทียบอ็อบเจกต์
`if`	เงื่อนไข
`else`	เงื่อนไขอื่น
`elif`	เงื่อนไขอื่น ๆ
`for`	การวนซ้ำ `for`
`while`	การวนซ้ำ `while`

คำสงวน	คำอธิบาย
`break`	หยุดการวนซ้ำ
`continue`	ดำเนินการต่อไป
`def`	การกำหนดฟังก์ชัน
`return`	คืนค่า
`class`	การกำหนดคลาส
`try`	การทดสอบ
`except`	การจัดการข้อผิดพลาด
`finally`	การทำงานท้ายสุด
`import`	การนำเข้าโมดูล
`from`	การนำเข้าบางส่วนจากโมดูล
`as`	การกำหนดชื่อใหม่ให้กับโมดูล
`lambda`	ฟังก์ชันแบบไม่มีชื่อ
`pass`	การผ่าน

หากมีการนำคำสงวนเหล่านี้มาตั้งเป็นชื่อตัวแปรหรือฟังก์ชันจะเกิดข้อผิดพลาดทางไวยากรณ์ (syntax error) เช่น

```
if = 10  # จะเกิด SyntaxError เนื่องจาก if เป็นคำสงวน
def = "hello"  # จะเกิด SyntaxError เนื่องจาก def เป็นคำสงวน
```

2.6 การควบคุมทิศทางการทำงานของโปรแกรม

โดยทั่วไปแล้วการควบคุมทิศทางการทำงานของโปรแกรมแบ่งออกเป็น 3 ชนิด หลัก ๆ คือ

1. **คำสั่งทำงานแบบตามลำดับ (Sequential statement)** โปรแกรมจะทำงานตามลำดับของคำสั่งที่กำหนดจากบรรทัดบนลงล่าง

ลองมาดูผังงาน (Flowchart) ของโปรแกรมสำหรับคำนวณหาพื้นที่สี่เหลี่ยมผืนผ้าที่ได้จากการนำค่าตัวแปร `width` และ `height` มาคูณกัน แล้วนำผลลัพธ์ที่ได้เก็บไว้ในตัวแปร `area` จากนั้นก็นำค่าที่อยู่ในตัวแปร `area` ออกมาแสดงผล

> **Flowchart คืออะไร**
>
> Flowchart หรือ ผังงาน เป็นเครื่องมือที่ใช้ในการแสดงขั้นตอน กระบวนการ หรือ ระบบ ในรูปแบบที่เข้าใจง่ายและเป็นระบบ ซึ่งมักใช้สัญลักษณ์ รูปร่าง และ เส้นเชื่อม ในการแสดงรูปแบบการดำเนินงานหรือขั้นตอนต่าง ๆ ของโปรแกรม