SE-ED COMPUTER BOOK developer

เขียนโปรแกรมด้วยภาษา Python ฉบับเพิ่มเติม กับ PyQt และ Pygame

SE-ED
inspiration starts here

สำหรับผู้ที่มีพื้นฐานการเขียนโค้ด Python มาแล้ว เพื่อเสริมทักษะการเรียนรู้ในระดับการใช้งานจริง

- **PyQt :** สร้าง UI ที่สวยงามด้วยวิดเจ็ตที่หลากหลาย
- **QtSql :** ชุดเครื่องมือจัดการฐานข้อมูลในแบบ CRUD
- **Pygame :** สร้างเกมแบบ 2D พร้อมตัวช่วยมากมาย

DOWNLOAD

ดาวน์โหลดโค้ดหนังสือเล่มนี้ได้ที่
http://www.developerthai.com

บัญชา ปะสีละเตสัง

ผู้เขียนหนังสือขายดีระดับ Best Seller ด้าน Programming

ซีเอ็ด

เขียนโปรแกรมด้วยภาษา Python ฉบับเพิ่มเติม กับ PyQt และ Pygame

โดย บัญชา ปะสีละเตสัง

สงวนลิขสิทธิ์ตามกฎหมาย โดย บัญชา ปะสีละเตสัง © พ.ศ. 2565
ห้ามคัดลอก ลอกเลียน ดัดแปลง ทำซ้ำ จัดพิมพ์ หรือกระทำอื่นใด โดยวิธีการใดๆ ในรูปแบบใดๆ
ไม่ว่าส่วนหนึ่งส่วนใดของหนังสือเล่มนี้ เพื่อเผยแพร่ในสื่อทุกประเภท หรือเพื่อวัตถุประสงค์ใดๆ
นอกจากจะได้รับอนุญาต

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

บัญชา ปะสีละเตสัง.

เขียนโปรแกรมด้วยภาษา Python ฉบับเพิ่มเติม กับ PyQt และ Pygame.-- กรุงเทพฯ : ซีเอ็ดยูเคชั่น, 2565.
336 หน้า.

1. ไพธอน (ภาษาคอมพิวเตอร์). 2. การเขียนโปรแกรม (คอมพิวเตอร์).
I. ชื่อเรื่อง.

005.133

Barcode (e-book) : 9786160846252

จัดพิมพ์และจัดจำหน่ายโดย

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)
SE-EDUCATION PUBLIC COMPANY LIMITED

เลขที่ 1858/87-90 ถนนเทพรัตน แขวงบางนาใต้ เขตบางนา กรุงเทพฯ 10260
โทรศัพท์ 0-2826-8000

[หากมีคำแนะนำหรือติชม ติดต่อที่ comment@se-ed.com]

SE-ED
inspiration starts here

คำนำ

ปัจจุบันภาษาไพธอนถูกนำไปใช้เป็นเครื่องมือในการเขียนโค้ดสำหรับงานแขนงต่าง ๆ ในรูปแบบที่หลากหลาย อย่างไรก็ตาม ถึงแม้ภาษาไพธอนจะมีวิธีการเขียนโค้ดที่เรียบง่าย แต่เมื่อนำไปพัฒนาแอปพลิเคชันที่ซับซ้อนมากขึ้น ความยุ่งยากและซับซ้อนในการเขียนก็เพิ่มขึ้นมาเป็นลำดับ ดังนั้น เพื่อเป็นการลดความยุ่งยากตามที่กล่าวมา จึงมีกลุ่มนักพัฒนาได้สร้างเครื่องมือในรูปแบบของไลบรารีเสริม เข้ามาช่วยให้การพัฒนาแอปพลิเคชันบางอย่างสามารถทำได้ง่ายขึ้น โดยมีไลบรารีให้เลือกใช้งานอย่างหลากหลายและมีวัตถุประสงค์ที่แตกต่างกันออกไป

SEED
inspiration starts here

สำหรับในหนังสือเล่มนี้ จะกล่าวถึงไลบรารี 2 อย่างที่กำลังได้รับความนิยมในกลุ่มนักพัฒนาที่ใช้ภาษาไพธอน นั่นก็คือ PyQt ที่เน้นการสร้าง GUI เพื่อติดต่อกับผู้ใช้ในแบบกราฟิก เป็นหลัก โดยมีวิดเจ็ตให้เลือกใช้งานอย่างหลากหลายและครอบคลุมเกือบทุกด้าน และไลบรารีอีกอันที่จะกล่าวถึงก็คือ Pygame ที่ใช้ในการสร้างเกมแบบ 2D พร้อมตัวช่วยมากมายซึ่งจะช่วยลดภาระความยุ่งยากในการเขียนโค้ดของเกมให้ง่ายขึ้น จนเราสามารถพัฒนาเกมได้เองโดยใช้ความรู้ในภาษาไพธอนแค่ระดับพื้นฐานทั่วไปก็เพียงพอแล้ว

ไลบรารีทั้งหมดที่จะกล่าวถึงในหนังสือเล่มนี้ต่างก็เรียนรู้ได้ง่าย และสามารถนำไปใช้งานได้จริง ดังนั้น ผู้เขียนจึงหวังว่า ผู้อ่านจะได้รับประโยชน์จากหนังสือเล่มนี้อย่างเต็มที่และครบถ้วน จนสามารถนำไปประยุกต์ใช้งานเพื่อสร้างแอปพลิเคชันด้านต่าง ๆ ได้ตามต้องการ

บัญชา ปะสีละเตสัง

banchar_pa@yahoo.com
facebook.com/DeveloperThai


SE-ED
inspiration starts here

สารบัญ

SE-ED inspiration starts here

SE-ED
inspiration starts here

SE-ED
inspiration starts here

SE-ED
inspiration starts here

SE-ED
inspiration starts here

SE-ED
inspiration starts here

1

พื้นฐาน PyQt และการจัดโครงร่าง

การพัฒนาแอปพลิชันในปัจจุบัน เรามักคำนึงถึงความสะดวกสบายของผู้ใช้งานเป็นหลัก ดังนั้น เรามักเลือกสร้างในรูปแบบที่มีการติดต่อกับผู้ใช้ในกราฟิก หรือที่เรียกว่า Graphic User Interface (GUI) แต่ในส่วนของภาษาไพธอน จะไม่มีเครื่องมือในการ GUI โดยตรง หรือถ้ามีก็เป็นพวกไลบรารีเสริมจากภายนอก เช่น TKinter แต่ชุดเครื่องมือดังกล่าวยังมีข้อบกพร่องหลายประการ นอกจากนี้ยังขาดความสวยงาม และความยืดหยุ่นต่อการพัฒนาแอปพลิเคชันที่ซับซ้อนอีกด้วย ดังนั้น จึงมีนักพัฒนาหลายกลุ่ม ได้พัฒนาชุดไลบรารีสำหรับการสร้าง GUI ที่มีความสามารถเหนือกว่า TKinter แม้จะให้เลือกใช้งานอยู่หลายอัน แต่ที่ได้รับความนิยมสูงสุดก็คือ PyQt ตามที่จะนำเสนอในหนังสือเล่มนี้ แต่อย่างไรก็ตาม ขอบเขตเนื้อหาของ PyQt มีค่อนข้างมาก โดยในบทนี้ เราจะเริ่มต้นจากลักษณะพื้นฐานและการจัดโครงร่างของ UI กันไปก่อน ส่วนวิธีการอื่นๆ จะกล่าวเพิ่มเติมในบทต่อๆ ไป

SE-ED
Inspiration starts here

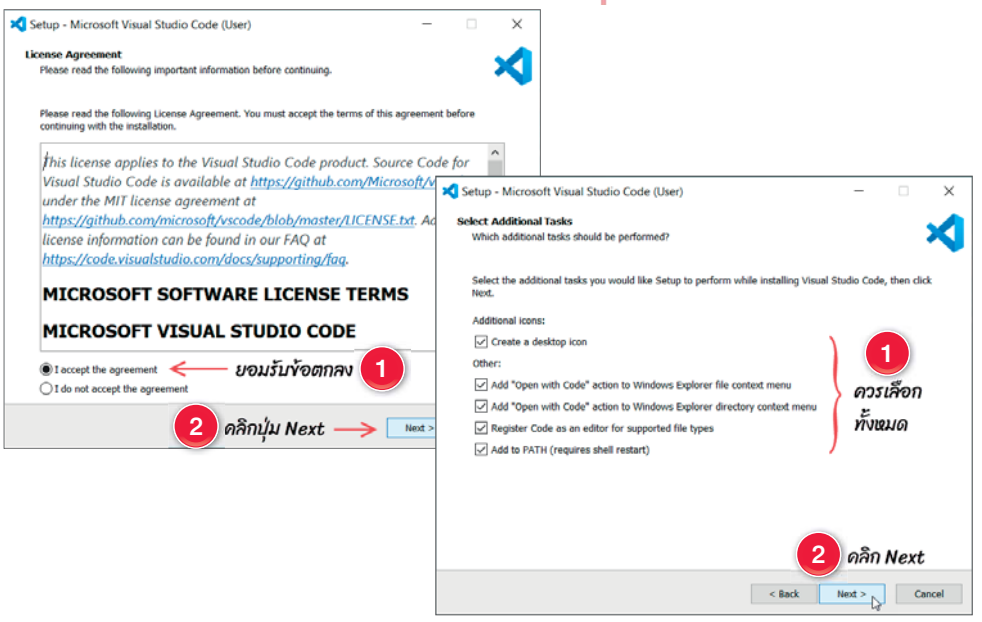
การจัดเตรียมเครื่องมือสำหรับเขียนโค้ด

ก่อนที่เราจะเข้าสู่เนื้อหาของการเขียนโค้ดตามที่จะกล่าวถึงในหัวข้อต่อๆ ไป สิ่งที่เราควรดำเนินการเป็นลำดับแรกก็คือ การจัดเตรียมเครื่องมือการเขียนและทดสอบโค้ด ซึ่งในหนังสือเล่มนี้ ผู้เขียนเลือกใช้ Visual Studio Code (VS Code) ร่วมกับ Jupyter Notebook เช่นเดียวกับในหนังสือ *การเขียนโปรแกรม Python ฉบับพื้นฐาน* แต่อย่างไรก็ตาม ในที่นี้ก็จะกล่าวทบทวนสิ่งที่ต้องจัดเตรียมโดยสังเขปอีกครั้ง ดังนี้

- การติดตั้งภาษาไพธอน
  - สามารถดาวน์โหลดได้ที่ https://www.python.org/downloads
  - ขั้นตอนการติดตั้งก็ทำเหมือนกับโปรแกรมทั่วไป

- การติดตั้ง VS Code
  - ดาวน์โหลดไฟล์ติดตั้งได้ที่ https://code.visualstudio.com/download
  - ขั้นตอนที่สำคัญในการติดตั้งเป็นดังภาพ

SE-ED
inspiration starts here

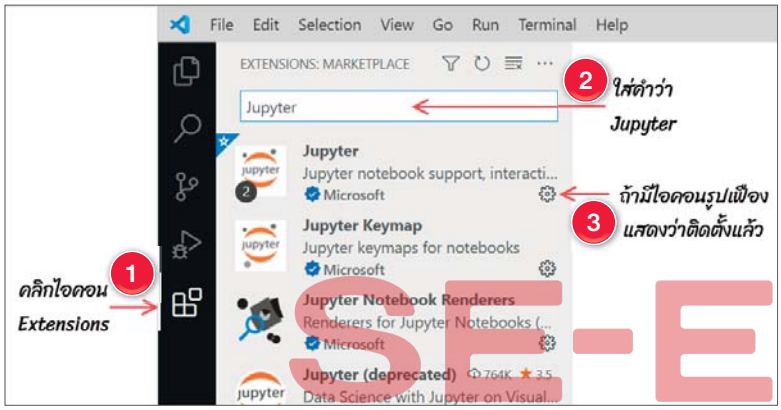


- การติดตั้ง Jupyter Notebook

  1) ที่ VS Code ให้คลิกที่ไอคอน Extension ที่แถบด้านซ้าย

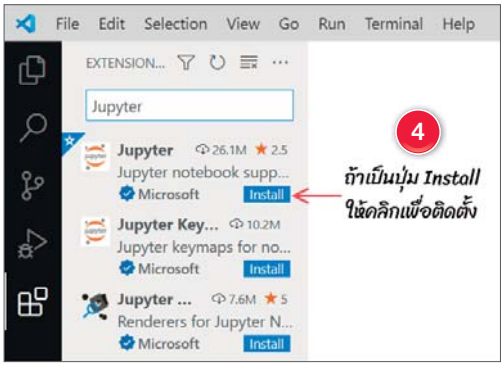
  2) พิมพ์คำว่า Jupyter ลงในช่องค้นหา (ขณะนั้นต้องเชื่อมต่ออินเทอร์เน็ต)

  3) ตามปกติ ผลการค้นหาจะมีรายการที่ชื่อ Jupyter (ที่ผ่านการตรวจสอบจาก Microsoft แล้ว) และถ้ามีไอคอนรูปเฟืองปรากฏที่กรอบรายการดังกล่าว แสดงว่า Jupyter ถูกติดตั้งเอาไว้แล้ว ซึ่งเราไม่จำเป็นต้องทำอะไรต่อ สามารถคลิกไปที่ไอคอน Extension เพื่อปิดหน้าจอการค้นหาส่วนเสริมได้เลย



SE-ED
inspiration starts here

  4) แต่หากที่กรอบรายการ Jupyter (ผลการค้นหา) ปรากฏปุ่มที่มีคำว่า Install แสดงว่ายังไม่ได้ติดตั้ง Jupyter ก็ให้เราคลิกปุ่มดังกล่าวเพื่อติดตั้งลงใน VS Code ได้เลย



ในหนังสือเล่มนี้ จะเก็บโค้ดทั้งหมดไว้ที่ **c:\python-plus** และเน้นการเขียนโค้ดด้วย Jupyter Notebook เป็นหลัก เพื่อความสะดวกต่อการรันทดสอบ และลดความยุ่งยากในการสร้างไฟล์แยกย่อยจำนวนมากสำหรับแต่ละตัวอย่าง ดังนั้น จึงขอกล่าวทบทวนแนวทางการเขียนและรันทดสอบโค้ดบน Jupyter Notebook ไว้พอสังเขป ดังนี้

1. ในที่นี้จะเก็บไฟล์ทั้งหมดของหนังสือเล่มนี้ไว้ที่ c:\python-plus ถ้ายังไม่มีโฟลเดอร์ดังกล่าว เราต้องสร้างขึ้นมาก่อน (ใช้วิธีใดก็ได้)
2. เข้าสู่ VS Code แล้วเปิดโฟลเดอร์ c:\python-plus เช่น เลือกเมนู File > Open Folder
3. หลังจากเปิดโฟลเดอร์ ถ้ายังไม่มีไฟล์ เราต้องสร้างไฟล์ขึ้นมาก่อน หลังจากนั้นให้ทำดังนี้

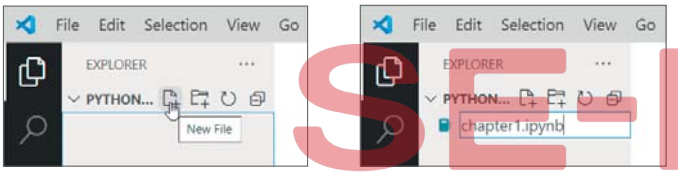
   1) ให้คลิกไอคอน New File

   2) กำหนดชื่อไฟล์ตามต้องการ โดยให้มีส่วนขยายเป็น .ipynb โดยไฟล์จะถูกจัดเก็บลงในโฟลเดอร์ที่เรากำลังเปิดในขณะนั้น สำหรับในหนังสือจะใช้ชื่อไฟล์ตามชื่อบท เช่น

      โค้ดทั้งหมดของบทที่ 1 จะเก็บไว้ในไฟล์ chapter1.ipynb

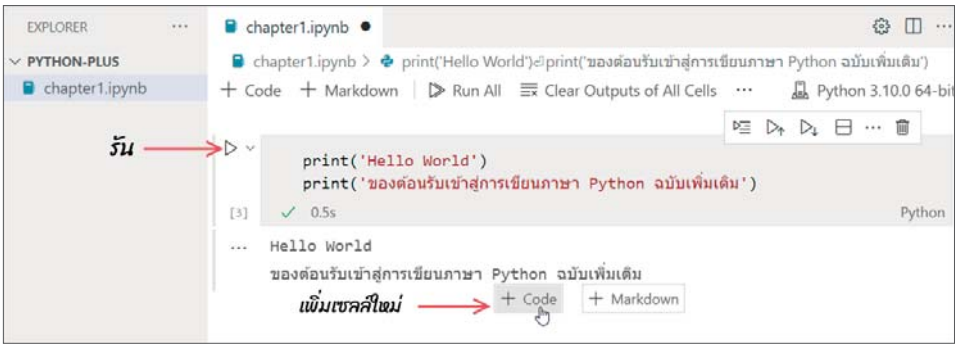
      โค้ดทั้งหมดของบทที่ 2 จะเก็บไว้ในไฟล์ chapter2.ipynb

      ...



SE-ED
inspiration starts here

   3) เมื่อเราคลิกเปิดไฟล์ที่มีส่วนขยายเป็น .ipynb ส่วน Editor ของ VS Code ก็จะแสดงเครื่องมือในมุมมองของ Jupyter โดยอัตโนมัติ จากนั้นก็เขียนโค้ดลงในช่องเซลล์ และรันทดสอบโดยคลิกไอคอนที่ขอบด้านซ้ายของช่องเซลล์ หรือกด <Ctrl + Alt + Enter>



สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับ VS Code และ Jupyter Notebook รวมถึงลักษณะปลีกย่อยอื่นๆ สามารถย้อนกลับไปทบทวนได้จากหนังสือ *การเขียนโปรแกรมด้วย Python ฉบับพื้นฐาน* ของผู้เขียน

ทบทวนหลักการสำคัญของคลาส

การเขียนโค้ดแทบทุกบทในหนังสือเล่มนี้ จะต้องเขียนรูปแบบของคลาสทั้งหมด แม้จะไม่มีองค์ประกอบที่ซับซ้อนมากนัก โดยใช้เพียงหลักการพื้นฐานธรรมดา แต่ผู้อ่านบางท่านอาจไม่ค่อยสันทัดเกี่ยวกับเรื่องนี้มากนัก ดังนั้น จึงขอนำลักษณะที่เราจะต้องใช้งานอยู่บ่อยๆ มากล่าวทบทวนอีกครั้งหนึ่ง ดังนี้

- ส่วนใหญ่แล้ว คลาสที่เราสร้างขึ้น จะเป็นการสืบทอดจากคลาสต้นแบบอันใดอันหนึ่ง ตามวัตถุประสงค์ของการใช้งาน ซึ่งลักษณะโครงสร้างจะเป็นดังนี้

```
class MyClass(Superclass):          #ถ้า Superclass คือชื่อคลาสต้นแบบที่จะสืบทอด
    def __init__(self):             #Initializer หรือเรียกสั้นๆ ว่า Init
        super().__init()__          #super() ใช้อ้างถึงคลาสต้นแบบ
        ...                         #super().__init__() จึงเป็นการอ้างถึง
                                    #Init ของคลาสต้นแบบ เพื่อการเชื่อมโยงถึงกัน
```

SEED
inspiration starts here

- ภายในคลาส ส่วนใหญ่เราจะสร้างสมาชิกเป็นเมธอด ซึ่งข้อกำหนดที่สำคัญคือ ต้องให้พารามิเตอร์ตัวแรกเป็นคำว่า self เพื่อใช้อ้างถึงสมาชิกหรือเมธอดอื่นๆ ในคลาสนั้น เช่น

```
class MyClass(Superclass):
    def __init__(self):
        super().__init()__

    def m1(self):
        pass

    def m2(self, a, b):
        self.m1()      #เรียกเมธอด m1() ผ่าน self
        ...
```

- เมื่อเราต้องการใช้คลาสที่สร้างขึ้น ก็ต้องกำหนดตัวแปรอ้างอิงหรืออินสแตนซ์ พร้อมกับสร้างออบเจ็กต์ของคลาสดังกล่าว ต่อไปก็เข้าถึงสมาชิกของคลาสผ่านอินสแตนซ์ เช่น

```
mycls = MyClass()
mycls.m1()
a = mycls.m2()
```

สำหรับลักษณะปลีกย่อยอื่นๆ หากอยู่เกินขอบเขตที่เราเคยได้เรียนรู้มา ก็จะอธิบายหลักการเพิ่มเติมให้ทราบในขั้นตอนที่เกี่ยวข้องต่อไป

เกี่ยวกับ PyQt และการติดตั้ง

Qt เป็นชุดเครื่องมือ (Toolkit) หรือไลบรารี (Library) ซึ่งใช้ในการสร้าง GUI (Graphic User Interface) สำหรับภาษา C++ ที่สามารถทำงานข้ามแพลตฟอร์มทั้งระบบ Windows, MacOS, Linux และอื่นๆ ซึ่งผู้ริเริ่มโครงการของ Qt ก็คือ Eirik Chambe-Eng และ Haavard Nord ในปี 1991 ที่ประเทศนอร์เวย์ ในยุคเริ่มแรกก็ได้ก่อตั้งเป็นบริษัทชื่อ Trolltech จนในปัจจุบันได้กลายมาเป็น The Qt Company ซึ่งนอกจาก Qt จะประกอบด้วยชุดเครื่องมือสำหรับการสร้าง GUI แล้ว ยังครอบคลุมถึง ชุดเครื่องมือเกี่ยวกับ ฐานข้อมูล มัลติมีเดีย กราฟิก และอื่นๆ อีกมากมาย รวมถึงการทำงานร่วมกับอุปกรณ์เคลื่อนที่ (Mobile) อีกด้วย

SE-ED
inspiration starts here

ส่วน PyQt เป็นชุดเครื่องมือของภาษาไพธอน (หรืออาจเรียกว่า Framework) ที่ใช้ Qt เป็นฐานในการทำงาน หรือกล่าวได้ว่า PyQt ก็คือชุดไลบรารีที่ห่อหุ้ม (Wrap) อยู่รอบๆ C++ Qt Library นั่นเอง ทั้งนี้ PyQt ถูกพัฒนาโดยบริษัท Riverbank Computing Limited ในสหราชอาณาจักร ซึ่งในขณะที่เขียนหนังสือเล่มนี้ PyQt มีการปรับปรุงมาจนถึงเวอร์ชัน 6 แล้ว หรือเรียกว่า PyQt6 และแม้ว่าวัตถุประสงค์หลักของ PyQt จะใช้ในการสร้าง GUI คล้ายกับ Tkinter ที่ได้กล่าวไว้ในหนังสือ *การเขียนโปรแกรมด้วย Python ฉบับพื้นฐาน* แต่ก็มีจุดเด่นที่น่าสนใจหลายประการ เช่น

- รูปร่างหน้าตาของแต่ละวิดเจ็ต จะสวยงามและน่าใช้งานมากกว่าของ Tkinter
- มีจำนวนวิดเจ็ตให้เลือกใช้งานที่หลากหลายและครอบคลุมมากกว่า
- PyQt ประกอบด้วยคลาสย่อยๆ มากกว่า 440 คลาส และฟังก์ชันมากกว่า 4,000 ฟังก์ชัน (และอาจมีจำนวนเพิ่มขึ้นเรื่อยๆ ตามเวอร์ชัน)
- นอกจากวิดเจ็ตในการสร้าง GUI แล้ว ใน PyQt ยังมีเครื่องมือสำหรับงานด้านอื่นๆ อีกหลายอย่าง เช่น ฐานข้อมูล กราฟ มัลติมีเดีย เน็ตเวิร์ก กราฟิก และแอนิเมชัน เป็นต้น ดังนั้น จึงมีทางเลือกสำหรับการประยุกต์ใช้งานที่หลากหลายกว่า
- PyQt รองรับการใช้งานข้ามแพลตฟอร์มเช่นเดียวกับ Tkinter

อย่างไรก็ตาม PyQt ไม่ใช่องค์ประกอบพื้นฐานที่มีอยู่แล้วในภาษาไพธอน ซึ่งเราใช้งานได้เลยเหมือนกับ Tkinter แต่เป็นชุดเครื่องมือจากภายนอกที่เราต้องติดตั้งเพิ่มลงไปในรูปแบบของโมดูล จึงจะใช้งานได้ ดังขั้นตอนต่อไปนี้

1. ขณะนั้นต้องเชื่อมต่อกับอินเทอร์เน็ต
2. เปิดเข้าสู่ VS Code แล้วเปิดแสดง Terminal (เมนู Terminal > New Terminal) หรืออาจเปิด Command ของระบบก็ได้
3. พิมพ์คำสั่งลงใน Terminal เพื่อติดตั้ง PyQt6 ดังนี้ (เวอร์ชันขณะที่เขียนหนังสือเล่มนี้)

```
pip install pyqt6
```

```
C:\python-plus>pip install pyqt6
```

หลังจากติดตั้งลงไปแล้ว PyQt ก็จะเป็นส่วนหนึ่งของภาษาไพธอน ต่อไปเราก็สามารถใช้งานจากที่ใดก็ได้ภายในเครื่องนั้น

พื้นฐานการสร้าง GUI ด้วย PyQt

SE-ED
inspiration starts here

แม้ขั้นตอนการสร้าง GUI ของ PyQt จะแตกต่างจาก Tkinter แต่หากเราผ่านการเรียนรู้วิธีการของ Tkinter มาแล้ว ก็จะช่วยได้มาก เพราะสามารถนำแนวคิดบางส่วนมาใช้ได้ ซึ่งจะช่วยให้เราเข้าใจวิธีการได้ง่ายและรวดเร็วขึ้น โดยลักษณะโค้ดพื้นฐานที่สุดในการแสดงเพียงวินโดว่างๆ ที่ยังไม่เขียนแบบคลาส จะเป็นดังนี้

ตัวอย่าง 1-1 การสร้าง GUI ด้วย PyQt ที่มีเพียงวินโดว่างๆ

```
from PyQt6.QtWidgets import QApplication, QWidget

app = QApplication([])

window = QWidget()
window.show()

app.exec()
```

- from PyQt6.Xxx คลาสทั้งหมดของ PyQt6 จะอยู่ในโมดูลที่มีชื่อขึ้นต้นด้วยคำว่า PyQt6 ทั้งหมด ส่วน Xxx คือชื่อกลุ่มย่อยๆ เช่น QtWidgets ซึ่งก็ขึ้นกับคลาสที่เราจะนำมาใช้งานว่าจัดอยู่ในกลุ่มใด ทั้งนี้ เราจะได้ศึกษาเรียนรู้กันไปเรื่อยๆ
- import QXxx, QYyy เป็นการระบุคลาสที่เราจะนำมาใช้งาน โดยคลาสทั้งหมดของ PyQt จะมีชื่อขึ้นด้วยตัว Q ซึ่งก็มาจากคำว่า Qt นั่นเอง
- app = QApplication([]) เป็นการกำหนดอินสแตนซ์หรือสร้างออบเจ็กต์ของคลาส QApplication นั่นเอง ซึ่งการกำหนดลิสต์ว่าง ([]) เป็นอาร์กิวเมนต์ก็เพราะในที่นี้เราไม่ได้ผ่านค่าใดๆ เข้าไป จึงอาจกำหนดเป็นลิสต์ว่างแทนก็ได้
- window = QWidget() เป็นการสร้างออบเจ็กต์ของคลาส QWidget เพื่อใช้เป็นวินโดวหรือฟอร์มในการจัดวางวิดเจ็ตอื่นๆ ทั้งหมดใน PyQt
- window.show() ตามปกติ วินโดวจะถูกซ่อนเอาไว้ก่อน ดังนั้น เราต้องเรียกขึ้นมาแสดงด้วย window.show() ทุกครั้ง จึงจะปรากฏวินโดวให้เห็น
- app.exec() เป็นการวนลูปเพื่อตรวจจับอีเวนต์ ในขณะที่กำลังรันแอปพลิเคชัน หรือเรียกว่า Event Loop ซึ่งก็เทียบเท่ากับเมธอด mainloop() ใน Tkinter นั่นเอง

SE-ED
inspiration starts here

หมายเหตุ

กรณีที่เราต้องนำเข้าหลายคลาสจากโมดูลเดียวกัน ซึ่งตามปกติเราสามารถใช้เครื่องหมาย * แทนคลาสหรืออื่นๆ ทั้งหมดในโมดูลนั้นได้ หรือเรียกว่า Wildcard Import เช่น

```
from PyQt6.QWidgets import *
```

แต่ทางผู้พัฒนา PyQt แนะนำให้หลีกเลี่ยงวิธีนี้ เพราะในแต่ละโมดูลจะประกอบด้วยตัวแปร ฟังก์ชัน และอื่นๆ อีกจำนวนมาก ซึ่งหากเราแทนด้วยเครื่องหมาย * จะหมายถึงองค์ประกอบทั้งหมดในโมดูลนั้น ซึ่งบางครั้งเราอาจประกาศชื่อตัวแปร หรือฟังก์ชัน หรืออื่นๆ ซ้ำกับสิ่งที่มีอยู่แล้วในโมดูล ก็อาจจะไปแทนที่การทำงานเดิมของสิ่งนั้น จนก่อให้เกิดข้อผิดพลาดได้

หากเราเขียนโค้ดดังกล่าวนี้ในแบบไฟล์ .py เมื่อรันโค้ดซ้ำๆ จะไม่เกิดปัญหาใดๆ แต่หากเราเขียนโค้ดด้วย Jupyter Notebook สมมติว่ารันครั้งแรกแล้วปิดวินโดว จากนั้นหากรันซ้ำครั้งต่อไป จะเกิดปัญหาโดยแจ้งข้อผิดพลาดว่า The kernel 'Python ...' died ... ก็ให้เราแก้ไขโดยเพิ่มโค้ดในขั้นตอนการสร้างอินสแตนซ์ของแอป ดังนี้

ตัวอย่าง 1-2 แนวทางการแก้ไขปัญหาการรันซ้ำเมื่อเขียนโค้ดบน Jupyter Notebook

```
from PyQt6.QtCore import QCoreApplication
from PyQt6.QtWidgets import QApplication, QWidget

#แก้ปัญหา Kernel died ใน Jupyter เมื่อรันซ้ำครั้งต่อไป
app = QCoreApplication.instance()
if app is None:
     app = QApplication([])

window = QWidget()
window.show()
app.exec()
```

อย่างไรก็ตาม การสร้าง GUI ที่ถูกต้องตามหลักการของ PyQt และสามารถจัดวางวิดเจ็ตลงไปได้ ต้องทำในรูปแบบของคลาสที่สืบทอดมาจาก QWidget (ส่วนใหญ่นิยมใช้คลาสนี้) โดยองค์ประกอบพื้นฐานที่ต้องมีภายในคลาสคือ Init พร้อมเรียก __init()__ ของ Superclass ดังแนวทางต่อไปนี้

SE-ED
inspiration starts here

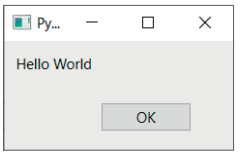
ตัวอย่าง 1-3 การสร้าง GUI ที่ถูกต้องตามหลักการของ PyQt

```
from PyQt6.QtCore import QCoreApplication
from PyQt6.QtWidgets import QApplication, QWidget

class MainWindow(QWidget):          #ใช้ชื่อคลาสอะไรก็ได้ แต่ให้สืบทอดจาก QWidget
     def __init__(self):
          super().__init__()        #หรือ super(MainWindow, self).__init__()

          self.setWindowTitle('PyQt GUI')    #ข้อความที่ Titlebar
          #self.show()                       #อาจเรียก show() ที่นี่ก็ได้

app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()               #สร้างอินสแตนซ์ของคลาสที่ใช้กำหนดวินโดว
window.show()                       #ถ้าเรียกเมธอดนี้ในคลาสแล้ว ก็ไม่ต้องเรียกที่นี่อีก
app.exec()
```

ต่อไป ถ้าเราจะสร้าง UI ก็วางวิดเจ็ตในคลาสของวินโดวได้เลย แต่เนื่องจากเรายังไม่ได้ศึกษาเกี่ยวกับการจัดโครงร่าง จึงจะใช้วิธีการระบุตำแหน่งที่จะวางมุมบนซ้ายของวิดเจ็ตโดยใช้เมธอด move(x, y) ซึ่งก็เทียบเท่ากับเมธอด place( ) ของ Tkinter นั่นเอง ดังแนวทางในตัวอย่างโค้ดถัดไป (รายละเอียดของวิดเจ็ตแต่ละชนิดจะกล่าวถึงในบทต่อไป แต่ในบทนี้จะใช้ Button และ Label ประกอบการอธิบายล่วงหน้าไปก่อน)

ตัวอย่าง 1-4 พื้นฐานการจัดวางวิดเจ็ตลงบนวินโดวด้วยการระบุตำแหน่งโดยตรง

```
from PyQt6.QtCore import QCoreApplication
from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('PyQt')

        lbl = QLabel('Hello World', self)    #ให้วางบนวินโดวนี้ (self)
        lbl.move(10, 10)

        btn = QPushButton('OK', self)
        btn.move(80, 50)

app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()
window.show()
app.exec()
```



SE-ED
inspiration starts here

ทั้งที่กล่าวมาในหัวข้อนี้ เป็นเพียงแนวทางการสร้าง GUI ในเบื้องต้นเท่านั้น ส่วนรูปแบบที่ซับซ้อนยิ่งขึ้น เราจะได้เรียนรู้ต่อไปเรื่อย ๆ

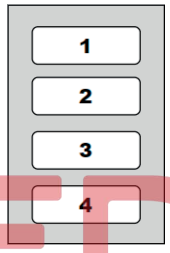
การกำหนดโครงร่างแบบต่าง ๆ

ถึงแม้เราสามารถจัดวางวิดเจ็ตโดยใช้เมธอด move( ) ดังที่ได้กล่าวไว้ในหัวข้อที่แล้ว แต่การกำหนดตำแหน่งอาจเป็นเรื่องที่ยุ่งยากพอสมควร ทั้งนี้ หาก UI ที่ต้องการออกแบบ สามารถจัดวางโดยใช้หลักการเดียวกันทั้งหมด เช่น เรียงจากซ้ายไปขวาในแถวเดียวกัน หรือเรียงจากบน

ลงล่างในแนวคอลัมน์เดียวกัน หรือจัดเรียงแบบช่องตาราง (กริด) เป็นต้น ก็อาจลดความยุ่งยากในการระบุตำแหน่ง โดยเปลี่ยนมาใช้วิธีจัดโครงร่าง (Layout) แบบใดแบบหนึ่งที่มีอยู่ใน PyQt ซึ่งอันที่น่าสนใจและนิยมใช้กันเป็นส่วนใหญ่คือ VBox Layout, HBox Layout และ Grid Layout หรืออาจใช้โครงร่างหลายแบบร่วมกัน เพื่อสร้าง UI ที่ซับซ้อนขึ้น สำหรับรายละเอียดของโครงร่างแต่ละแบบมีดังนี้

โครงร่างแบบ VBox

VBox มาจากคำว่า Vertical Box หรือถ้าเรียกตามชื่อคลาสก็คือ QVBoxLayout เป็นการจัดเรียงวิดเจ็ตในแนวตั้งจากบนลงล่าง หรือในคอลัมน์เดียวกัน ซึ่งก็คล้ายกับเมธอด pack(side=TOP) ของ Tkinter นั่นเอง โดยมีแนวทางดังต่อไปนี้



- โครงร่างแบบนี้ กำหนดด้วยคลาส QVBoxLayout ซึ่งอยู่ในโมดูล PyQt6.QtWidgets
- เราต้องสร้างอินสแตนซ์ของคลาส QVBoxLayout แล้วกำหนดให้เป็นโครงร่างของวินโดวด้วยเมธอด setLayout(vBox) ภายในคลาสที่เราใช้สร้างวินโดวนั้น เช่น

SE-ED
inspiration starts here

```
class MainWindow(QWidget):
    ...
    vBox = QVBoxLayout()
    self.setLayout(vBox)    #กำหนดให้เป็นโครงร่างของวินโดวนี้
```

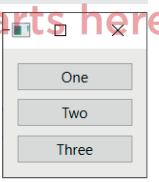
- ต่อไปก็กำหนดวิดเจ็ตแต่ชนิดที่จะสร้าง UI แล้วเพิ่มลงในโครงร่างด้วยเมธอด addWidget() เช่น

```
lbl = QLabel('Test')
vBox.addWidget(lbl)          #เพิ่มวิดเจ็ตลงบนโครงร่าง
...
btn = QPushButton('OK')
vBox.addWidget(btn)
```

วิดเจ็ตจะถูกเพิ่มลงใน VBox โดยเรียงตามลำดับที่เราเรียกเมธอด addWidget() สำหรับขั้นตอนอื่นๆ ก็ทำเหมือนเดิม ดังแนวทางในตัวอย่างต่อไปนี้

ตัวอย่าง 1-5 การวางวิดเจ็ตลงในโครงร่างแบบ VBox

```
from PyQt6.QtCore import QCoreApplication
from PyQt6.QtWidgets import QApplication, QWidget, \
                            QVBoxLayout, QWidget, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()

        vBox = QVBoxLayout()
        self.setLayout(vBox)

        btn1 = QPushButton('One')
        vBox.addWidget(btn1)

        btn2 = QPushButton('Two')
        vBox.addWidget(btn2)

        btn3 = QPushButton('Three')
        vBox.addWidget(btn3)

app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()
window.show()
app.exec()
```



SE-ED
inspiration starts here

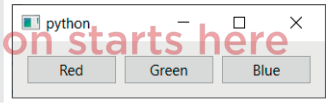
โครงร่างแบบ HBox

HBox มาจากคำว่า Horizontal Box หรือถ้าเรียกตามชื่อคลาสก็คือ QHBoxLayout เป็นการจัดเรียงวิดเจ็ตในแนวนอนจากซ้ายไปขวา หรือในแถวเดียวกัน ซึ่งก็คล้ายกับเมธอด pack(side=LEFT) ของ Tkinter นั่นเอง

สำหรับหลักการต่างๆ ก็คล้ายกับ VBox เพียงแต่เปลี่ยนมาใช้คลาส QHBoxLayout ดังแนวทางในตัวอย่างถัดไป ซึ่งเพื่อลดขั้นตอนการเขียนโค้ดที่ซ้ำซ้อนกัน ก็จะนำโค้ดในส่วนของการกำหนดและเพิ่มวิดเจ็ตลงใน HBox มาสร้างเป็นเมธอด แล้วผ่านข้อมูลที่จำเป็นของแต่ละวิดเจ็ตไปทางพารามิเตอร์

ตัวอย่าง 1-6 การวางวิดเจ็ตลงในโครงร่างแบบ HBox

```
from PyQt6.QtCore import QCoreApplication, QSize, Qt
from PyQt6.QtWidgets import QApplication, QHBoxLayout, \
                            QWidget, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        hBox = QHBoxLayout()
        self.setLayout(hBox)

        for b in ['Red', 'Green', 'Blue']:
            self.add_button(b, hBox)

    def add_button(self, text, layout):
        btn = QPushButton(text)
        layout.addWidget(btn)


app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()
window.show()
app.exec()
```



SE-ED
inspiration starts here

โครงร่างแบบ Grid

โครงร่างแบบกริด (Grid) เป็นการจัดเรียงวิดเจ็ตคล้ายกับช่องตาราง เช่นเดียวกับการใช้เมธอด grid() ของ Tkinter โดยเราต้องระบุลำดับแถวและคอลัมน์ดังในภาพถัดไป

หลักการพื้นฐานทั่วไปของ Grid จะคล้ายกับ VBox และ HBox ดังที่กล่าวมาแล้ว แต่ส่วนที่ต่างกันคือ เมธอด addWidget() ต้องระบุลำดับแถวและคอลัมน์ลงไปด้วย ในรูปแบบดังนี้

```
addWidget(widget, row, column)
```

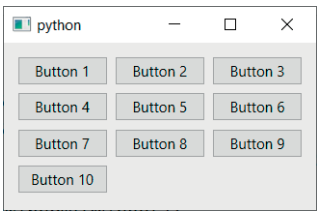
```
grid.addWidget(btn1, 0, 0)      #แถวที่ 1 คอลัมน์ที่ 1
grid.addWidget(btn2, 0, 1)      #แถวที่ 1 คอลัมน์ที่ 2
grid.addWidget(btn3, 1, 0)      #แถวที่ 2 คอลัมน์ที่ 1
```

0, 0	0, 1	0, 2
1, 0	1, 1	1, 2
2, 0	2, 1	2, 2

สำหรับแนวทางการเขียนโค้ดในตัวอย่างถัดไป เนื่องจากเป็นวิดเจ็ตชนิดเดียวกันลงใน Grid และต้องเขียนโค้ดในลักษณะเดิมซ้ำๆ กัน ดังนั้น จึงจะใช้ลูป for ในการจัดวางวิดเจ็ต โดยแยกบางขั้นตอนออกมาสร้างเป็นเมธอด เพื่อให้โค้ดสั้นลง

ตัวอย่าง 1-7 การวางวิดเจ็ตลงในโครงร่างแบบ Grid

```
from PyQt6.QtCore import QCoreApplication, QSize, Qt
from PyQt6.QtWidgets import QApplication, QGridLayout, \
                            QWidget, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        grid = QGridLayout()
        self.setLayout(grid)
        '''
        ลักษณะเลขลำดับของการจัดวางที่ต้องการ
        0 1 2
        3 4 5
        6 7 8
        '''

        r, c = 0, 0
        for i in range(10):
            if i % 3 == 0:          #ลำดับที่ 3 หารลงตัว
                r += 1              #ให้ขึ้นแถวใหม่
                c = 0               #กลับไปเริ่มที่คอลัมน์แรก

            self.add_button(i+1, r, c, grid)
            c += 1

    def add_button(self, num, row, col, layout):
        btn = QPushButton(f'Button {num}')
        layout.addWidget(btn, row, col)


app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()
window.show()
app.exec()
```

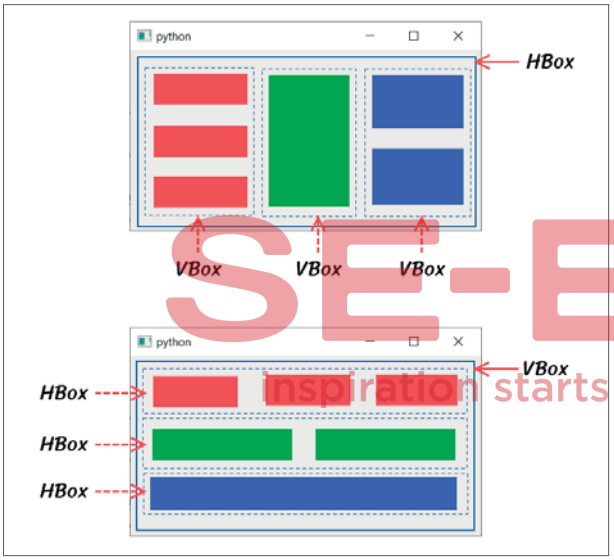


SE-ED
inspiration starts here

การจัดวางโครงร่างซ้อนกัน (Nested Layout)

โดยทั่วไปแล้ว หากเป็นการสร้างแอปพลิเคชันในระดับการใช้งานจริง ลักษณะการออกแบบ UI อาจมีความซับซ้อนเกินกว่าที่จะใช้โครงร่างเพียงแบบเดียวได้ ดังนั้น หากเราไม่ใช้เมธอด move() ก็อาจต้องใช้โครงร่างหลายๆ แบบร่วมกัน เช่น VBox ร่วมกับ HBox หรือ VBox ร่วมกับ Grid หรือใช้ทั้งหมดร่วมกันก็ได้ ทั้งนี้ก็ขึ้นกับลักษณะ UI ที่เราต้องการเป็นหลัก ซึ่งขอให้ดูจากแนวทางต่อไปนี้ เพื่อเลือกใช้โครงร่างที่จะวางซ้อนกัน

- อันดับแรก เราต้องพิจารณาลักษณะ UI ที่เราต้องการก่อนว่า ควรใช้โครงร่างแบบใดซ้อนกันจึงจะเหมาะสมที่สุด หากใช้ได้หลายแบบ ควรเลือกแบบที่ซับซ้อนน้อยที่สุด



SE-ED
Inspiration starts here

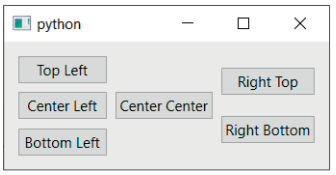
- ในขั้นตอนการเขียนโค้ด หากจะใช้แนวทางของผู้เขียน ก็อาจเริ่มจากการกำหนดโครงร่างชั้นนอกสุดขึ้นมาก่อน จากนั้นก็ค่อยกำหนดโครงร่างชั้นในพร้อมกับวางวิดเจ็ตลงไป ทั้งนี้ คลาสของโครงร่างแต่ละแบบ จะมีเมธอดชื่อ **addLayout()** สำหรับใช้ในการวางโครงร่างย่อยๆ ซ้อนลงไป คล้ายกับการวางวิดเจ็ต เช่น

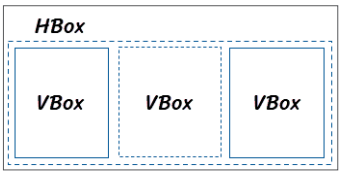
```
#สมมติว่าโครงร่างชั้นนอกจะให้เป็น HBox ชั้นในจะใช้ VBox (มีหลายอัน)
hBox = QHBoxLayout()
self.setLayout(vBox)          #วางโครงร่างชั้นนอกลงในวินโดว

vBox1 = QVBoxLayput()         #กำหนดโครงร่างชั้นในตามปกติ
hBox.addLayout(vBox1)         #วางโครงร่างชั้นในลงในโครงร่างชั้นนอก
...
```

```
vBox2 = QVBoxLayput()          #กำหนดโครงร่างชั้นในอันถัดไป
hBox.addLayout(vBox2)          #วางโครงร่างชั้นในอันถัดไปลงในโครงร่างชั้นนอก
...
```

สำหรับแนวทางการใช้งานโดยรวม ขอให้ดูจากตัวอย่างโค้ดต่อไปนี้

ตัวอย่าง 1-8 การวางโครงร่างซ้อนกัน โดยชั้นนอกจะใช้ HBox ส่วนชั้นในจะใช้ VBox





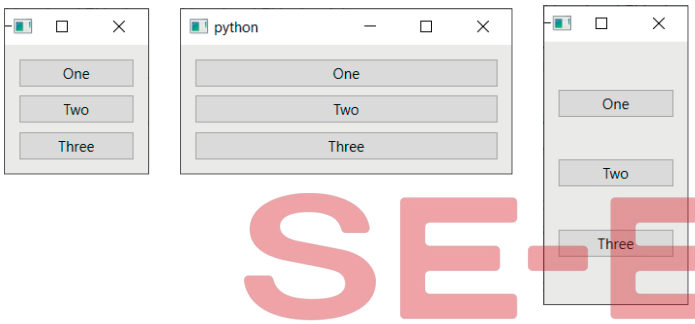

```
from PyQt6.QtCore import QCoreApplication
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, \
                            QHBoxLayout, QVBoxLayout

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        main_layout = QHBoxLayout()      #ถ้าใช้ Grid จะซับซ้อนกว่านี้
        self.setLayout(main_layout)

        inner_layout1 = QVBoxLayout()
        main_layout.addLayout(inner_layout1)
        btn1 = QPushButton('Top Left')
        inner_layout1.addWidget(btn1)
        inner_layout1.addWidget(QPushButton('Center Left'))
        inner_layout1.addWidget(QPushButton('Bottom Left'))

        inner_layout2 = QVBoxLayout()
        main_layout.addLayout(inner_layout2)
        inner_layout2.addWidget(QPushButton('Center Center'))

        inner_layout3 = QVBoxLayout()
        main_layout.addLayout(inner_layout3)
        inner_layout3.addWidget(QPushButton('Right Top'))
        inner_layout3.addWidget(QPushButton('Right Bottom'))

app = QCoreApplication.instance()
if app is None: app = QApplication([])
```

SE-ED
inspiration starts here

```
window = MainWindow()
window.show()
app.exec()
```

การกำหนดขนาดและระยะห่าง

การสร้าง UI ที่ผ่านมานั้น เราไม่ได้กำหนดขนาดและระยะห่างระหว่างวิดเจ็ต หรือเป็นการใช้ค่าดีฟอลต์นั่นเอง ซึ่งในบางกรณี UI ที่ได้อาจไม่ตรงตามที่เราต้องการอย่างแท้จริง นอกจากนี้ ก็มีปัญหาบางอย่างที่เราต้องคำนึงถึงก็คือ วิดเจ็ตบางชนิดอาจเปลี่ยนขนาดและระยะห่างตามการย่อหรือขยายขนาดของวินโดว ดังภาพ (ใช้โครงร่างแบบ VBox)



SE-ED
inspiration starts here

เพื่อป้องกันการแสดงผล UI ที่ผิดเพี้ยนดังที่กล่าวมา เราควรกำหนดขนาด และ/หรือ ระยะห่างที่แน่นอนให้แก่วิดเจ็ต โดยมีแนวทางดังต่อไปนี้

การกำหนดขนาดของวิดเจ็ต

การกำหนดขนาดของวิดเจ็ตให้คงที่ เราต้องใช้เมธอดอันใดอันหนึ่งดังต่อไปนี้

setFixedSize(size)	กำหนดขนาดวิดเจ็ตให้มีค่าคงที่ (ไม่สามารถย่อหรือขยายขนาดได้)
setFixedWidth(width)	กำหนดความกว้างของวิดเจ็ตให้มีค่าคงที่ (ส่วนความสูงใช้ค่าดีฟอลต์)
setFixedHeight(height)	กำหนดความสูงของวิดเจ็ตให้มีค่าคงที่ (ส่วนความกว้างใช้ค่าดีฟอลต์)

สำหรับแนวทางการใช้งาน มีหลักการโดยสังเขปดังนี้

- หากใช้เมธอด setFixedSize เราต้องกำหนดทั้งความกว้างและความสูงด้วยคลาส QSize ซึ่งคลาสนี้อยู่ในโมดูล PyQt6.QtCore โดยสร้างอินสแตนซ์ของคลาสนี้พร้อมระบุความกว้างและสูงที่ต้องการ จากนั้นก็นำไปกำหนดให้แก่เมธอด setFixedSize( ) เช่น

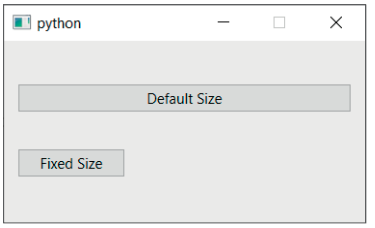
```
size = QSize(50, 30)
btn.setFixedSize(size)
#หรือเขียนแบบรวบรัดเป็น
#btn.setFixedSize(QSize(50, 30))
```

- ถ้าใช้เมธอด setFixedWidth() หรือ setFixedHeight() สามารถกำหนดค่าเป็นตัวเลขตามที่ต้องการได้เลย แต่เมธอดทั้งสองอันนี้ ไม่มีอยู่ใน QWidget จึงใช้กำหนดขนาดของวินโดวไม่ได้ (กรณีนี้ให้ใช้ setFixedSize แทน)

สำหรับแนวทางการใช้งาน ให้ดูจากโค้ดตัวอย่างต่อไปนี้

ตัวอย่าง 1-9 เป็นการเปรียบเทียบระหว่างการใช้ขนาดค่าดีฟอลต์ (ไม่ระบุ) กับขนาดคงที่

```
from PyQt6.QtCore import QCoreApplication, QSize
from PyQt6.QtWidgets import QApplication, QVBoxLayout, \
                            QWidget, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setFixedSize(QSize(300, 150))
        vBox = QVBoxLayout()
        self.setLayout(vBox)

        btn1 = QPushButton('Default Size')
        vBox.addWidget(btn1)

        btn2 = QPushButton('Fixed Size')
        btn2.setFixedWidth(90)
        vBox.addWidget(btn2)

app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()
window.show()
app.exec()
```



SE-ED
inspiration starts here

การกำหนดระยะห่างและจัดแนวขอบของวิดเจ็ต

คลาสในกลุ่มการจัดโครงร่าง เช่น VBox จะมีเมธอด setSpacing() ในการกำหนดระยะห่างหรือช่องว่างระหว่างวิดเจ็ต แต่หากเราใช้เมธอดนี้เพียงอย่างเดียว จะไม่ได้ผลตามต้องการ ซึ่งจากการทดสอบของผู้เขียนพบว่า วิธีที่ได้ผลคือ ต้องใช้ร่วมกับเมธอด setAlignment() เพื่อกำหนดว่า จะจัดแนวของวิดเจ็ตชนิดให้ชิดขอบด้านใดของโครงร่าง ซึ่งมีวิธีการโดยสังเขปดังนี้

- การจัดแนวขอบของวิดเจ็ต จะต้องใช้เมธอด setAlignment() แล้วกำหนดค่าจากตัวเลือกของคลาส Qt เช่น หากต้องให้วิดเจ็ตทั้งหมดใน VBox ชิดขอบด้านบนของโครงร่าง ก็กำหนดโค้ดเป็น

```
vBox.setAlignment(Qt.AlignmentFlag.AlignTop)  #Top, Left, Right, Bottom
```

- หลังจากนั้น ก็กำหนดระยะห่างระหว่างวิดเจ็ตด้วยเมธอด setSpacing() โดยระบุค่าเป็นตัวเลขตามต้องการ เช่น vBox.setSpacing(7)

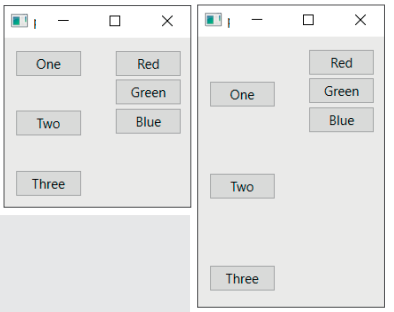
สำหรับแนวทางการใช้งาน ให้ดูจากโค้ดตัวอย่างต่อไปนี้

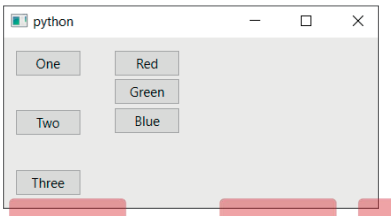
SE-ED
inspiration starts here

ตัวอย่าง 1-10 เป็นการเปรียบเทียบระหว่างการใช้ระยะห่างแบบค่าดีฟอลต์ (ไม่ระบุ) กับระยะห่างที่กำหนดค่าเอง

```
from PyQt6.QtCore import QCoreApplication, QSize, Qt
from PyQt6.QtWidgets import QApplication, QVBoxLayout, \
                            QHBoxLayout, QWidget, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        hBox = QHBoxLayout()
        hBox.setAlignment(Qt.AlignmentFlag.AlignLeft)
        hBox.setSpacing(30)
        self.setLayout(hBox)

        vBox1 = QVBoxLayout()
        hBox.addLayout(vBox1)
        for x in ['One', 'Two', 'Three']:
            self.add_button(x, vBox1)
```

```
            vBox2 = QVBoxLayout()
            vBox2.setAlignment
              Qt.AlignmentFlag.AlignTop)
            vBox2.setSpacing(2)
            hBox.addLayout(vBox2)
            for x in ['Red', 'Green', 'Blue']:
                 self.add_button(x, vBox2)

      def add_button(self, text, layout):
            btn = QPushButton(text)
            btn.setFixedWidth(60)
            layout.addWidget(btn)


app = QCoreApplication.instance()
if app is None: app = QApplication([])

window = MainWindow()
window.show()
app.exec()
```





SE-ED
Inspiration starts here

การจัดรูปแบบด้วย Qt Style Sheets (QSS)

ตามปกตินั้น ใน PyQt ก็มีคลาสสำหรับการปรับแต่งและกำหนดรูปแบบพื้นฐานทั่วไปอยู่แล้ว เช่น คลาส QFont, QColor เป็นต้น แต่สิ่งที่พิเศษอย่างหนึ่งก็คือ PyQt สามารถรองรับการจัดรูปแบบในลักษณะเดียวกับ CSS (Cascading Style Sheets) ที่เราใช้ในการปรับแต่งเพจ ดังนั้น เราอาจเรียกวิธีการนี้ว่า Qt Style Sheets (QSS) แต่อย่างไรก็ตาม QSS จะรองรับการปรับแต่งเฉพาะลักษณะพื้นฐานเพียงบางส่วนเท่านั้น ถึงกระนั้นก็ตาม QSS ก็ยังเป็นสิ่งที่น่าสนใจและนิยมใช้กันอย่างแพร่หลาย โดยมีรายละเอียดดังต่อไปนี้

พร็อปเพอร์ตี้พื้นฐานของ QSS

หากผู้อ่านเคยใช้ CSS ในการจัดรูปแบบเว็บเพจมาแล้ว ก็จะเข้าใจหลักการของ QSS ได้ทันที แต่บางคนอาจยังไม่เคยเรียนรู้สิ่งนี้มาก่อน ดังนั้น จึงขอแนะนำพื้นฐานบางส่วนให้ทราบไว้ล่วงหน้า โดยลักษณะแต่ละอย่างที่เราจะจัดรูปแบบ เช่น สีข้อความ พื้นหลัง เส้นขอบ ฟอนต์ จะเรียกว่าพร็อปเพอร์ตี้ (Property) ซึ่งใน QSS ก็รองรับการใช้พร็อปเพอร์ตี้ตามแบบของ CSS แต่ไม่ครบทั้งหมด สำหรับในหัวข้อนี้ จะแนะนำให้รู้จักกับพร็อปเพอร์ตี้ที่น่าสนใจบางส่วนซึ่งมักนำมาใช้งานอยู่บ่อย ๆ ดังนี้

color	กำหนดสีข้อความ ในแบบ • ชื่อสีในภาษาอังกฤษ เช่น red • rgb(r, g, b) โดยทั้ง r, g และ b จะมีค่าระหว่าง 0-255 • #hhhhhh กำหนดค่าแบบ hex เป็นเลข 0-9 หรือ a-f จำนวน 6 ตัว เช่น #00ffee
background-color หรือ **background**	ใช้กำหนดสีพื้นหลังในแบบเดียวกับ color เช่น background-color: yellow
border-width	ขนาดความหนาของเส้นขอบ เช่น border-width: 2px
border-style	รูปแบบของเส้นขอบ ซึ่งอาจเป็น solid, dashes หรือ dotted และต้องกำหนดความหนาเส้นขอบมากกว่า 0 พร็อปเพอร์ตี้นี้จึงจะมีผล เช่น border-style: solid
border-color	สีของเส้นขอบ กำหนดค่าเช่นเดียวกับ color โดยต้องกำหนดความหนาเส้นขอบมากกว่า 0 พร็อปเพอร์ตี้นี้จึงจะมีผล เช่น border-color: green
border	กำหนดทั้ง 3 ในพร็อปเพอร์ตี้เดียวกัน เช่น border: 2px solid blue
font-family	กำหนดชื่อฟอนต์ เช่น font-family: tahoma โดยต้องมีฟอนต์ดังกล่าวอยู่ในเครื่องนั้น
font-size	ขนาดของฟอนต์ เช่น font-size: 12pt
font-style	มักใช้เมื่อต้องการฟอนต์แบบตัวเอียง เช่น font-style: italic
font-weight	น้ำหนักของฟอนต์ เช่น font-weight: bold
height	ความสูงของวิดเจ็ต เช่น height: 50px
margin-top/left/right/bottom	ระยะห่างกับวิดเจ็ตถัดไป ของด้าน บน/ซ้าย/ขวา/ล่าง ตามลำดับ เช่น margin-top: 10px หรือ margin-right: 5px
margin	ระยะห่างกับวิดเจ็ตถัดไปทั้ง 4 ด้าน (เท่ากันหมด) เช่น margin: 8px
text-align	การจัดแนวข้อความภายในวิดเจ็ตว่าจะให้ชิดขอบด้านใดระหว่าง left, center, right เช่น text-align: center
width	ความกว้างของวิดเจ็ต เช่น width: 100px

SE-ED
inspiration starts here

พร็อปเพอร์ตี้ที่แสดงในตาราง เป็นคำแนะนำโดยสังเขปเท่านั้น หากผู้อ่านต้องการทราบรายละเอียดที่ชัดเจน สามารถดูได้จากหนังสือที่เกี่ยวกับ CSS เช่น หนังสือ *พัฒนา Web App แบบ Responsive ด้วย Bootstrap5 ร่วมกับ Vue.js และ Node.js* ของผู้เขียน หรือศึกษาเพิ่มเติมจากเว็บไซต์ที่เกี่ยวข้อง ส่วนวิธีการใช้งาน ให้ดูในหัวข้อถัดไป

หมายเหตุ

สำหรับการกำหนดพร็อปเพอร์ตี้ที่เกี่ยวกับสี เช่น color, background-color ก็ใช้หลักการเดียวกับ CSS หรือสามารถดูเพิ่มเติมได้จากเว็บไซต์ https://htmlcolorcodes.com

การกำหนดรูปแบบ QSS ให้กับวิดเจ็ตโดยตรง

วิดเจ็ตส่วนใหญ่ใน PyQt จะมีเมธอด setStyleSheet() สำหรับกำหนดรูปแบบ QSS ให้กับมันโดยตรง ดังรูปแบบต่อไปนี้

```
setStyleSheet('''
    property_1: value_1;
    property_2: value_2;
    ...
''')
```

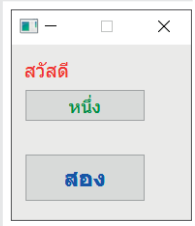
SE-ED
inspiration starts here

- ต้องคั่นระหว่างพร็อปเพอร์ตี้และค่าของมันด้วยเครื่องหมาย :
- ต้องคั่นระหว่างพร็อปเพอร์ตี้ด้วยเครื่องหมาย ;
- เราสามารถกำหนดพร็อปเพอร์ตี้จำนวนเท่าไหร่ก็ได้
- พร็อปเพอร์ตี้ทั้งหมดต้องเขียนรวมไว้ในสตริงเดียวกัน (นำสตริงย่อยมาเชื่อมต่อกันได้)

แต่อย่างไรก็ตาม ขอให้ระลึกไว้เสมอว่า **พร็อปเพอร์ตี้แต่ละอัน จะใช้ได้ผลกับวิดเจ็ตเพียงบางชนิดเท่านั้น** ซึ่งวิธีที่เราจะรู้ได้ก็คือ นำพร็อปเพอร์ตี้นั้นไปทดสอบกับวิดเจ็ตเป้าหมาย หากไม่มีการเปลี่ยนแปลง แสดงว่าพร็อปเพอร์ตี้ดังกล่าวใช้กับวิดเจ็ตนั้นไม่ได้ สำหรับแนวทางการนำไปใช้งาน มีดังนี้

ตัวอย่าง 1-11 พื้นฐานการจัดรูปแบบวิดเจ็ตด้วย QSS

```
from PyQt6.QtCore import QCoreApplication, QSize, Qt
from PyQt6.QtWidgets import QApplication, QWidget, \
                            QVBoxLayout, QPushButton, QLabel

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setFixedSize(QSize(250, 500))

        vBox = QVBoxLayout()
        self.setLayout(vBox)
        vBox.setAlignment(Qt.AlignmentFlag.AlignTop)

        lbl = QLabel('สวัสดี')
        lbl.setStyleSheet(
            '''
            color: red;
            font-family: 'Tahoma';
            font-size: 12pt;
            '''
        )
        vBox.addWidget(lbl)

        btn1 = QPushButton('หนึ่ง')
        btn1.setFixedWidth(100)
        btn1.setStyleSheet(
            '''
            color: blue;
            font-family: Tahoma;
            font-size: 11pt
            '''
        )
        vBox.addWidget(btn1)

        btn2 = QPushButton('สอง')
        btn2.setFixedSize(QSize(100, 60))
        btn2.setStyleSheet(
            '''
            color: green;
            font-family: Tahoma;
            font-size: 14pt;
            font-weight: bold;
            margin-top: 20px;
```



SE-ED
inspiration starts here

เขียนโปรแกรมด้วยภาษา Python ฉบับเพิ่มเติม กับ PyQt และ Pygame

ในหนังสือเล่มนี้ จะกล่าวถึงไลบรารี 2 อย่างที่กำลังได้รับความนิยมในกลุ่มนักพัฒนาที่ใช้ภาษาไพธอน นั่นก็คือ PyQt ที่เน้นการสร้าง GUI เพื่อติดต่อกับผู้ใช้ในแบบกราฟิกเป็นหลัก โดยมีวิดเจ็ตให้เลือกใช้งานอย่างหลากหลายและครอบคลุมเกือบทุกด้าน และไลบรารีอีกอันที่จะกล่าวถึงก็คือ Pygame ที่ใช้ในการสร้างเกมแบบ 2D พร้อมตัวช่วยมากมาย ซึ่งจะช่วยลดภาระความยุ่งยากในการเขียนโค้ดของเกมให้ง่ายขึ้น จนเราสามารถพัฒนาเกมเองได้โดยใช้ความรู้ในภาษาไพธอนแค่ระดับพื้นฐานทั่วไปก็เพียงพอแล้ว ไลบรารีทั้งหมดที่จะกล่าวถึงในหนังสือเล่มนี้ต่างก็เรียนรู้ได้ง่าย และสามารถนำไปใช้งานได้จริง ดังนั้น ผู้อ่านจะได้รับประโยชน์จากหนังสือเล่มนี้อย่างเต็มที่และครบถ้วน จนสามารถนำไปประยุกต์ใช้งานเพื่อสร้างแอปพลิเคชันด้านต่าง ๆ ได้ตามต้องการ

SE-ED
inspiration starts here

เนื้อหาในหนังสือประกอบด้วย :

www.se-ed.com sbc.fans SE-ED Publisher

พร้อมจำหน่ายในรูปแบบ
☑ e-book (PDF) ☐ audiobooks
☐ e-book (EPUB) ☐ audio CD / MP3
☑ ปกอ่อน ☐ LARGE PRINT (ตัวอักษรขนาดใหญ่)

ISBN 978-616-08-4625-2
249 บาท

คอมพิวเตอร์/การเขียนโปรแกรม