

เคล็ดลับการเพิ่มผลงาน ลดความซับซ้อนของงานด้วย Excel VBA

**โดย
สมเกียรติ ฟุ่งเกียรติ
Excel Expert Training**

www.ExcelExpertTraining.com

www.XLSiam.com

excel@XLSiam.com

02 318 7021, 02 718 9331

มีนาคม 2555

สงวนลิขสิทธิ์

สารบัญ

เคล็ดลับการเพิ่มผลงาน ลดความซับซ้อนของงานด้วย Excel VBA	3
ข้อควรคำนึงก่อนใช้ Excel VBA.....	5
อยากเก่ง VBA..ตามประสาคนไม่มีเวลา.. จะเริ่มอย่างไร.....	9
ขั้นตอนสร้างรหัส VBA โดยใช้ Macro Recorder.....	13
โครงสร้างชุดคำสั่ง VBA	17
วิธีใช้ VBA ให้คุ้มค่า.....	24
ทางออกที่ดีกว่าการใช้ VBA	30
เคล็ดลับการใช้ Macro Recorder	35
วิธีใช้ Macro Recorder ช่วยในการเรียนรู้ VBA	42
วิธีสร้างชุดคำสั่งควบคุมระบบเมื่อเปิดปิดแฟ้ม	48
รหัส VBA ที่ยืดหยุ่นตามการเปลี่ยนแปลงใน Excel	53
VBA กับการจัดการฐานข้อมูล.....	57
วิธีใช้รหัส VBA แบบ Evaluate ในการรับส่งข้อมูล	65
วิธีส่งข้อมูลไปยังเซลล์ภายในตารางที่กำหนด.....	69
วิธีส่งข้อมูลไปยังเซลล์รับข้อมูล ณ ตำแหน่งใดก็ได้	71
วิธีส่งข้อมูลจากตารางที่มีขนาดไม่แน่นอน.....	73
วิธีส่งข้อมูลไปบันทึกต่อท้ายรายการที่มีอยู่แล้ว	75
วิธีส่งข้อมูลกลับไปแก้รายการเดิมที่มีอยู่แล้ว	77
วิธีเลือกส่งข้อมูลตามลักษณะรายการเก่าใหม่	80
วิธีเรียกใช้รหัสเกี่ยวกับตาราง Excel	82
VBA กับการตัดสินใจ	86
วิธีสร้างสัญญาณเตือน...ภัย.....	92
วิธีสั่งให้ VBA ทำงานทวนซ้ำหลายรอบ	97
วิธีสร้างสูตรเพื่อนำมาใช้ในงานเฉพาะด้าน	103
รหัส VBA ที่น่าสนใจจาก www.Excel-VBA.com	107
รหัส VBA ที่น่าสนใจจาก www.MindSpring.com	112

เคล็ดลับการเพิ่มผลงาน ลดความซับซ้อนของงานด้วย Excel VBA

Excel VBA เป็นเครื่องมือสำคัญที่ต้องหาทางใช้ให้เป็น เพราะ Excel VBA จะช่วยควบคุม Excel ให้ทำงานเองตั้งแต่ต้นจนจบได้อย่างรวดเร็ว ไม่ผิดพลาดแม้งานนั้นๆมีหลายขั้นตอนหรือมีลำดับที่ซับซ้อน ทำให้มนุษย์เรามีความสุขในการทำงานมากขึ้น ช่วยประหยัดเวลาที่เสียไปกับการนั่งอยู่หน้าจคอมพิวเตอร์ ช่วยเพิ่มเวลาว่างสำหรับใช้ชีวิตส่วนตัว

เมื่อทราบประโยชน์ของ Excel VBA เช่นนี้แล้ว เราควรเรียนรู้หาทางนำ Excel VBA มาใช้อย่างเร็วที่สุด ไม่ต้องรอให้เก่ง Excel ก่อนก็ได้ ไม่ควรคิดว่า Excel VBA เป็นเรื่องยากหรือเป็นเครื่องมือขั้นสูงเก็บไว้ให้คนที่เก่ง Excel หรือต้องเป็นโปรแกรมเมอร์เท่านั้นไว้ใช้กัน

อย่างไรก็ตาม Excel VBA กลับกลายเป็นเครื่องมือที่ใช้กันน้อยมาก มักติดปัญหาตรงที่ว่า ไม่รู้จะเริ่มต้นเรียนรู้ Excel VBA กันอย่างไรดี หนังสือหนึ่งหาที่เป็นภาษาไทยก็มีอยู่แค่ไม่กี่เล่ม ครั้นหยิบหนังสือขึ้นมาพลิกดู พออ่านได้เพียงไม่กี่หน้าก็ต้องปิดทิ้ง รู้สึกหมดหวัง ยอมแพ้เพราะอ่านแล้วไม่รู้เรื่อง รู้สึกว่า Excel VBA ยากเหลือเกิน อะไรก็ไม่รู้ทำไมต้องมีรหัสยาวๆเต็มไปหมด กว่าจะอ่านให้หมดเล่มเราจะใช้เวลาที่ไหนมาอ่านกัน เวลาที่มีอยู่นั้นก็ต้องใช้ทำงานประจำเต็มวันอยู่แล้ว แม้รู้ยู่่ว่า Excel VBA สามารถช่วยเพิ่มเวลาว่างให้ชีวิต แต่ตอนนี้เวลาที่มีอยู่ก็เหลือเวลาว่างให้เรียนรู้ Excel VBA ได้น้อยเหลือเกิน

เคล็ดลับการเพิ่มผลงาน ลดความซับซ้อนของงานด้วย Excel VBA เป็นเนื้อหาสั้นๆ มุ่งให้คนทำงานซึ่งไม่จำเป็นต้องเป็นโปรแกรมเมอร์ สามารถใช้เวลาว่างที่มีอยู่น้อยนิดนั้น มาเรียนรู้สิ่งใหม่ๆที่จะสร้างประโยชน์ได้คุ้มกับค่าของเวลาที่เสียไป เคล็ดลับไม่กี่อย่างซึ่งจะนำมาแนะนำต่อไปนั้น จะช่วยทำให้รู้สึกว่ Excel VBA ไม่ใช่เรื่องยากอีกต่อไป

วิธีใช้ Excel VBA ในเคล็ดลับการเพิ่มผลงาน ลดความซับซ้อนของงานด้วย Excel VBA มีหลายส่วนแตกต่างจากที่ทราบกัน หรือแม้แต่อาจถึงขั้นขัดแย้งกับวิธีที่ยึดถือกันมา ก็ต้องขอให้เตรียมตัวเตรียมใจไว้ก่อน อะไรที่ผู้อื่นบอกว่าสำคัญ แต่ที่นี้อาจเห็นว่าไม่

สำคัญ ไม่ต้องใช้เลยก็เป็นได้ เพราะถ้ามัวใช้เวลาไปกับการเรียนรู้ Excel VBA ทีละขั้น หรือเสียเวลาใส่ใจในทุกประเด็นตามแบบการเขียนโปรแกรมที่ดี เห็นงานที่ต้องทำส่งเจ้านายคงทำส่งไม่ทันตามกำหนดเป็นแน่

VB vs VBA vs Excel VBA vs VBE vs Macro

VB ย่อมาจาก Visual Basic เป็นโปรแกรมหนึ่งซึ่งใช้ควบคุมการทำงานของ Windows

VBA ย่อมาจาก Visual Basic for Applications เป็นส่วนหนึ่งของ VB

Excel VBA เป็นส่วนหนึ่งของ VBA ซึ่งใช้กับโปรแกรม Microsoft Excel โดยเฉพาะ ถ้านั่งสอใช้คำว่า VBA จะเป็นโปรแกรมที่ใช้กว้างๆกับ Application ต่างๆของ Microsoft Windows เช่น Word, PowerPoint, Access, Excel ดังนั้นถ้าอยากหาหนังสือ VBA มาศึกษาเพิ่มเติม ต้องมีคำว่า Excel VBA เพื่อใช้เฉพาะเจาะจงกับ Excel

VBE เป็นตัวโปรแกรม Visual Basic Editor ซึ่งติดมาให้ใช้พร้อมกับโปรแกรม Excel เราใช้ VBE เป็นโปรแกรมสำหรับสร้าง แก้ไข และสามารถสั่งงานให้รหัสคำสั่งของ VBA ทำงาน

Macro เป็นชื่อเก่าใช้กันมานานตั้งแต่สมัยโปรแกรม Lotus 1-2-3 และ Excel รุ่นแรกๆ ซึ่งใช้ Macro เก็บรหัสคำสั่งตามคำสั่งบนเมนู ซึ่งพบว่ามีข้อจำกัดอยู่มาก เพราะ Macro จะใช้ทำงานได้ในขอบเขตเท่าที่มีคำสั่งบนเมนูให้ใช้เท่านั้น ต่อมา Microsoft จึงดัดแปลง VB เป็น VBA ให้สามารถทำงานได้มากกว่าที่ Macro ทำได้ เช่น การสั่งให้ Excel ทำงานทวนซ้ำหลายๆรอบ ต้องอาศัยคำสั่งของ VBA

โดยทั่วไป Macro กับ VBA เป็นสิ่งที่สร้างผลงานได้เช่นเดียวกัน อาจถือว่า Macro เป็นส่วนหนึ่งของ VBA ไปแล้วก็ได้ ซึ่งในตัว Excel มีคำสั่งเมนู **Developer > Record Macro** ทำหน้าที่สร้างรหัส VBA บันทึกตามลำดับการสั่งงานจากเมาส์และแป้นพิมพ์ ในกรณีนี้รหัส Macro ที่ได้จากการบันทึกจึงกลายเป็นรหัส VBA นั่นเอง

(เมนู Developer จะปรากฏให้เห็นต่อเมื่อสั่ง File > Options > Customize Ribbon แล้วกาเลือกเมนูชื่อ Developer ใน Main Tab ที่แสดงด้านขวาของจอ)

ข้อควรคำนึงก่อนใช้ Excel VBA

ผู้ใช้ Excel ส่วนมากมักคิดว่า อะไรก็ตาม ปัญหาใดก็ตามที่คิดไม่ออก หาทางใช้ Excel แบบธรรมดาไม่ได้ ต้องใช้ Excel VBA ช่วยหาคำตอบได้แน่ เชื่อกันว่า Excel VBA ต้องเป็นทางออกสุดท้าย แต่แล้วพอใช้ Excel VBA ได้สักพัก กลับพบว่าวิธีการที่ตนเลือกใช้ กลายเป็นปัญหาต่อเนื่องไม่รู้จบ ครั้นจะคิดถอยหลัง จะเลิกใช้ Excel VBA กลับไปใช้วิธีการเหมือนเดิมก็สายไปเสียแล้ว ดังนั้นจึงขอให้คำนึงในประเด็นต่อไปนี้ ก่อนที่จะคิดใช้ Excel VBA (จากนี้ไปขอเรียก Excel VBA ย่อๆว่า VBA)

รู้จัก Excel ดีพอหรือยัง

ก่อนคิดจะหันไปใช้ VBA เป็นเครื่องมือหาคำตอบวิธีใหม่ เราควรรู้จักตัวเครื่องมือที่มีมากมายหลากหลายชนิดให้ครบถ้วนก่อน ทั้งเครื่องมือที่เป็นคำสั่งบนเมนู สูตรคำนวณ และวิธีออกแบบตารางที่ถูกต้อง โดยไม่จำเป็นต้องรู้สึกถึงขั้นใช้เป็นทุกอย่าง เพียงแต่ขอให้รอบรู้ว่สิ่งที Excel มีเตรียมไว้ให้ใช้พร้อมอยู่แล้วนั้น สามารถนำไปใช้กับงานประเภทใดได้บ้าง โดยเฉพาะคำสั่งบนเมนูประเภทที่ใช้ทำงานได้อัตโนมัติ

ข้อผิดพลาดที่พบเห็นเสมอ เกิดจากการขาดความรู้ว่า Excel สามารถใช้ทำงานอะไรได้บ้าง ทำให้หันไปวนววยวางแผนใช้ VBA ซึ่งกว่าจะสร้างรหัส VBA ได้ครบถ้วน ก็ต้องลองผิดลองถูกเสียเวลาและเหนื่อยยากมิใช่น้อย โดยหารู้ไม่ว่าแค่คลิกเดียวบนเมนู หรือแค่จัดโครงสร้างตารางใหม่ให้ถูกต้อง ก็สามารถสร้างงานที่ต้องการได้เช่นกัน

ขอให้พึงระลึกไว้เสมอว่า ถ้าเป็นงานพื้นที่เป็นปัญหาประจำของทุกคน ทางบริษัท Microsoft ได้จัดเตรียมเครื่องมือใน Excel ไว้เรียบร้อยแล้ว โดยไม่จำเป็นต้องชวนขวยไปใช้ VBA ให้เสียเวลาหรอก เพียงแต่เราอาจต้องใช้คำสั่งบนเมนูหลายอย่างนำมาใช้ร่วมกันให้เป็น

คุ้มไหม ใช้ซ้ำบ่อยไหม

ในประเด็นเรื่องคุ้มไหมใช้แค่คุ้มเรื่องเงินทอง ในแง่ประโยชน์ที่สามารถประเมินเป็นตัวเงินได้จากงานที่ใช้ VBA เท่านั้น แต่ควรคำนึงถึงค่าเสียเวลาของตัวเองด้วย ถ้าต้องใช้เวลาลึก 50 ชั่วโมง (สมมติว่าใช้เวลาสร้างงาน 25 วัน วันละ 2 ชั่วโมง) แล้วค่าแรงของตัวเองไปเข้าไปชั่วโมงละ 1,000 บาท ดังนั้นแฟ้มงานที่ใช้ VBA ย่อมมีต้นทุนถึง 50,000 บาททีเดียว จากนั้นยังต้องคอยกลับมาเสียเวลานับปีในการปรับปรุงแก้ไข ซึ่งย่อมตึกกลับเป็นต้นทุนเพิ่มเข้าไปได้อีก บางแฟ้มกว่าจะเสร็จสมบูรณ์อาจมีค่าที่นึกไม่ถึงสูงเป็นแสนเป็นล้านบาท

สิ่งที่นำมาเปรียบเทียบกับต้นทุนที่เสียไป ได้แก่ ระบบงานง่ายขึ้น ช่วยลดความซับซ้อน ลดความซ้ำซ้อน ลดโอกาสที่จะเกิดข้อผิดพลาดได้ไหม ถ้าต้องทำงานนั้นซ้ำแล้วซ้ำอีกจะช่วยประหยัดเวลาทำงานได้วันละกี่ชั่วโมง ช่วยลดความกังวล ความเครียด ความห่วงใย ความกลัวว่า งานที่สร้างขึ้นจะหมดข้อผิดพลาด สามารถส่งงานได้ทันกำหนดแน่นอนหรือไม่

มีเวลาไหม ตามเทคโนโลยีทันหรือไม่

กว่าจะเริ่มใช้ VBA เป็น คุณต้องลงทุนลงแรงเสียเวลาอ่านหนังสือ แบ่งเวลาทำงานมาเข้าอบรม แต่เวลาที่เสียไปนี้เป็นเพียงช่วยให้คุณเริ่มรู้จักว่า VBA เป็นอย่างไร มีประโยชน์อย่างไรได้เท่านั้น ถ้าอยากใช้ VBA ให้เป็นชนิดที่เรียกตนเองว่าโปรแกรมเมอร์ จะต้องเสียเวลาให้กับ VBA อีกมากมาย เพราะยังมีหนังสือเล่มหนาให้อ่านอีกเป็นตั้ง ดังนั้นหากคุณมีเวลาเพียงน้อยนิดในแต่ละวันให้กับ VBA ก็ขอให้ตั้งหลักไว้ว่า รอให้เกิดปัญหาขึ้นก่อนเถอะแล้วค่อยนึกถึง VBA ประเภททำไปแก้ไขไปที่ละเล็กละน้อย ไม่จำเป็นต้องเสร็จแบบสมบูรณ์เพียบพร้อม หรือใช้รหัสที่สั้นกะทัดรัดที่สุด หรือจำเป็นต้องเสียเวลาไปกับการหาทางทำให้รหัส VBA ทำงานเร็วที่สุด

นอกจากนั้นแม้ว่าตัวรหัส VBA ในวันนี้ส่วนใหญ่เป็นตัวรหัสที่ใช้กันมาตั้งแต่ต้น แต่ทุกครั้งที่มีการ Excel รุ่นใหม่ จะพบว่ามีรหัสใหม่เพิ่มขึ้นเสมอ บางครั้งยังปรับเปลี่ยนวิธีใช้รหัสใหม่ๆ ต่างไปจากเดิม ซึ่งถือเป็นหน้าที่ของผู้ใช้ VBA จะต้องคอยติดตามข่าวสารการปรับปรุงให้ทันยุคทันสมัย แล้วกลับไปแก้ไขรหัสเก่าๆ ให้ใช้ทำงานต่อไปได้ตลอด

มีคนแทนหรือไม่

จะไม่ใช่ปัญหาเลยถ้างานที่คุณทำนั้นไม่เกี่ยวข้องกับคนอื่น คุณจะหยุดหรือคุณจะทำออกไป ไม่กระทบกับระบบงานส่วนรวม แต่ถ้าเกี่ยวข้องกับงานส่วนรวม ย่อมเกิดปัญหาต่อเนื่องร้ายแรงอย่างมากต่อบริษัท

ลองนึกดูว่า ถ้าแฟ้มงานที่ใช้ VBA ช่วยนั้นเกิดไม่ทำงานตามเดิมขึ้นมา แต่ไม่สามารถหาใครมาช่วยแก้ไขได้ทันเวลา ผู้ใช้งานที่ติดนิสัยทำงานแบบสบาย พอเปิดแฟ้มขึ้นมาแล้วกดปุ่มเดียวสั่งให้ Excel ทำงานแบบอัตโนมัติ กลับพบว่าแฟ้มไม่ทำงานเองแบบเดิมเสียแล้ว งานจะหยุดเดิน ครั้นจะหันกลับไปใช้ระบบงานแบบเดิมก็ลืมกันไปหมดแล้ว

ผู้บริหารหรือหัวหน้าควรคำนึงถึงประเด็นตัวตายตัวแทนนี้ให้ดี ก่อนที่จะยอมให้พนักงานหรือลูกน้องคนใดใช้ VBA จะต้องหาตัวแทนเข้าไปประกบ ร่วมสร้างงาน ยิ่งถ้างานใดมีความสำคัญต่อความอยู่รอดของบริษัท จำต้องกำหนดให้เขียนคู่มืออธิบายรหัสทุกชั้นที่ใช้ไว้เสมอ

ระบบเริ่มตายตัว รู้จักปัญหาครบถ้วนหรือยัง

ตัวโครงสร้างของรหัส VBA และตัวรหัส VBA จะถูกแก้ไขเปลี่ยนแปลงได้โดยมนุษย์เข้าไปเขียนแก้ไขใน VBE เท่านั้น ตัวรหัส VBA โดยทั่วไปไม่สามารถปรับเปลี่ยนตามโครงสร้างตารางในชีทที่เปลี่ยนไป ดังนั้นก่อนที่จะคิดใช้ VBA ควรรองจนกว่าระบบงานตายตัว ชัดเจน และไม่มีปัญหาใหม่เกิดขึ้นตามมาอีก เพราะหากต้องย้อนกลับไปแก้ไขรหัสที่เขียนไว้เมื่อปีก่อน คงยากที่ใครจะจำได้ว่า จะต้องกลับไปแก้ไขอะไรที่ส่วนไหนของรหัสนับร้อยนับพันบรรทัด

จงจำไว้ว่า สักวันหนึ่งคุณจะจำรหัส VBA ที่เขียนไว้ไม่ได้ ดังนั้นขอให้แยกรหัสออกเป็นชุดย่อยๆ และใช้สั่งให้ทำงานเพียงขั้นเดียว หรือเป็นชุดคำสั่งที่ใช้ควบคุมการทำงานเพียงระบบเดียวที่เกี่ยวข้องกันชัดเจนเท่านั้น เพื่อช่วยทำให้ย้อนกลับไปแก้ไขได้ง่าย และถ้าเกิดข้อผิดพลาดขึ้นมา ย่อมถูกจำกัดให้ส่งผลกระทบกับระบบงานหนึ่งๆ เท่านั้น

หมดทางอื่นแล้วหรือยัง

หากคุณคิดแล้วว่า ยังไงก็ต้องหาทางใช้ VBA ให้ได้ ทั้งๆที่ตัวเองไม่ได้เก่ง Excel เท่าไฉนนัก หน่วยงานก็เล็กๆ ไม่มีตัวตายตัวแทนอย่างที่เคยเอนไว้ข้างต้นนั้นหรอก ปัญหา ก็ยังไม่แน่มันอน อาจเกิดปัญหาต้องจัดระบบใหม่อีกเมื่อใดก็ได้ แต่จะขอใช้ VBA เพื่อ ช่วยลดขั้นตอนของงานลงบ้าง สิ่งใดที่ต้องทำซ้ำแล้วซ้ำอีกจะได้ใช้ VBA ควบคุมให้ ทำงานเองตั้งแต่ต้นจนจบ หากคิดว่าหมดทางออกอื่นๆแล้ว ก็ขอให้ใช้ VBA กันอย่าง ฉลาด รหัสใดที่ยากนักให้ตัดทิ้ง รหัสใดไม่จำเป็น ก็ไม่ต้องใช้ เพื่อทำให้ง่ายต่อการ แก้ไขในภายหลัง

อยากเก่ง VBA..ตามประกาศคนไม่มีเวลา.. จะเริ่มอย่างไร

ข้อจำกัดเรื่องเวลาเป็นสิ่งที่สำคัญที่สุดของคนที่ยืนอยู่กับการทำงาน ทำงานตั้งแต่เช้าจนเย็นเลย ไปจนถึงค่ำ กลับบ้านแล้วยังเอางานติดตัวกลับไปทำต่อเสียอีก ไหนจะต้องแบ่งเวลาไปประชุมในตอนกลางวัน เขียนรายงานในตอนกลางคืน นานๆที่จะได้ใช้เวลาช่วงเสาร์ อาทิตย์ไปเที่ยวพักผ่อนกับครอบครัว โอกาสที่จะได้เรียนรู้สิ่งใหม่นี้น้อยมาก จะมีช่วงที่ได้ไปฝึกอบรมนั่นแหละที่ได้เปิดหูเปิดตาไปกับเขาบ้าง เรื่องที่ว่าให้อ่านหนังสือตำรา วิธีใช้ VBA เล่มโตตั้งแต่หน้าแรกจนหน้าสุดท้าย ... เมินเสียเถอะ ...ไม่มีเวลา

ปัญหาประเภทนี้เป็นปัญหาเหมือนถามว่า ไก่กับไข่อะไรเกิดก่อนกัน เพราะว่าถ้าไม่มีเวลาเรียนรู้ VBA นั้น เนื่องจากทำงานจนหมดเวลาอยู่แล้ว แต่ถ้าอยากมีเวลาเหลือเพิ่มเติมละ ต้องหาเวลาใช้ VBA ให้เป็น ถ้าใช้ VBA เป็น งานจะเสร็จเร็วขึ้น ทำให้มีเวลาวางไปทำอย่างอื่น ติดอยู่ตรงที่ว่าตอนนี้ไม่มีเวลาเหลือ จะเริ่มต้นหาทางเก่ง VBA ได้อย่างไร

คนเป็นที่พึ่งแห่งตน

จุดเริ่มต้นก่อนที่จะเก่ง VBA ได้ ต้องขึ้นกับตัวคุณเองต้องช่วยตัวเองให้เป็นก่อน อย่าคิดพึ่งคนอื่นให้ยกรหัส VBA มาให้คุณใช้ เพราะหากวันหนึ่งรหัส VBA นั้นเกิดหยุดไม่ทำงานได้เช่นเคย แล้วคุณหาทางแก้ไขเองไม่เป็น คุณจะต้องกลับไปง้อขอรับรองให้คนอื่นช่วยแก้ไขให้เรื่อยไป

ขอให้ตั้งใจสร้างงานขึ้นด้วยฝีมือของตนเอง ได้แค่ไหนก็แค่นั้น แคหาทางทำให้ Excel ทำงานได้เร็วกว่าเดิมก็ชื่นใจแล้ว ถ้าทำได้เอง คุณจะเกิดความภาคภูมิใจ เกิดความมั่นใจ แล้วจะหาทางพัฒนาฝีมือต่อไปได้เอง

เริ่มจากง่ายไปยาก

กว่าจะหาทางทำให้ VBA ทำงานได้อัตโนมัติตั้งใจ ไม่ใช่เรื่องง่าย ไม่เหมือนการใช้คำสั่งบนเมนูของ Excel ที่คลิกแล้วก็จะทำงานได้ทันที เนื่องจากรหัส VBA ไม่สามารถ

เกิดขึ้นได้เอง แต่ต้องอาศัยฝีมือของคนสร้างรหัส VBA ขึ้นมาใช้ ซึ่งตัวรหัสอาจเกิดจากการใช้ Macro Recorder หรือต้องเขียนเอง

ในช่วงเริ่มต้น ขอแนะนำให้เรียนรู้ VBA จากการใช้ Macro Recorder ช่วยสร้างรหัส VBA ขึ้นมา แล้วเรียนรู้รหัสคำสั่งต่างๆที่เกิดขึ้น โดยเริ่มจากการใช้ Macro Recorder บันทึกขั้นตอนการทำงานง่ายๆสั้นๆขึ้นมาแกระหัส พอเริ่มคุ้นเคยและเห็นว่ารหัสสามารถทำงานได้ดังใจแล้ว จึงค่อยๆเข้าไปแก้ไขดัดแปลงรหัสให้ต่างไปจากเดิม ลองผิดลองถูก ถ้าทำผิดก็ขอให้จำไว้ว่าผิดเพราะอะไร แม้รหัส VBA ที่ดัดแปลง จะทำงานต่างจากที่คิดไว้ ขอให้ถือเป็นบทเรียนเพื่อจำไว้เสมอ เพราะเราสามารถนำรหัสนั้นกลับมาใช้ในต่างวาระต่างวัตถุประสงค์ ซึ่งย่อมเกิดประโยชน์ได้เช่นกัน

อย่าใจร้อนสร้างรหัสยาวๆ หรือพิถีพิถันกับตัวโครงสร้างรหัสให้ตรงกับหลักที่กำหนดไว้ในหนังสือ แต่ขอให้มุ่งสร้างรหัสสั้นๆที่สามารถทำงานเสร็จทันเวลาตามต้องการให้ได้ก่อน ส่วนตัวโครงสร้างรหัสยังไม่สมบูรณ์นักก็อย่างพึงใส่ใจ

เรียนรู้จาก VBA Help

ทราบไหมว่า หนังสือ VBA ที่เห็นวางขายกันเกลื่อนนั้น มีน้อยเล่มที่เขียนเองทุกตัวอักษร หนังสือส่วนใหญ่ใช้ข้อมูลที่ลอกมาจาก VBA Help ซึ่งติดมากับโปรแกรม Microsoft Office แล้วนำมาดัดแปลงแก้ไขข้อความ จัดเรียงบทใหม่ แก่ตัวอย่างไม่ให้เข้ากับ VBA Help

ถ้าอยากอ่าน VBA Help ให้เปิด Excel ขึ้นมาก่อน จากนั้นกดปุ่ม **ALT+F11** หรือใช้คำสั่งเมนู **Developer > Visual Basic** เพื่อเปิดโปรแกรม VBE ขึ้นมา แล้วกดปุ่ม **F1** หรือใช้เมนู **Help** ขึ้นมาศึกษา ซึ่งถ้าสงสัยรหัสตัวใดก็ให้คลิกที่รหัสตัวที่เขียนไว้นั้น แล้วกดปุ่ม F1 จะเปิด Help เฉพาะเรื่องรหัสนั้นขึ้นมาให้เห็นโดยตรง

นอกจากนี้เรายังสามารถค้นหาแฟ้ม Help ที่มีนามสกุล .chm โดยไม่ต้องเปิดผ่าน VBE ทั้งนี้ต้องติดตั้ง Office แบบสมบูรณ์ก่อนจึงจะพบแฟ้มดังกล่าว ให้ดับเบิลคลิกที่ชื่อแฟ้มเพื่อเปิด Help

ซื้อหนังสือเก็บไว้หลายเล่ม หรือค้นหาจากเว็บ

เนื่องจากใน VBA Help มีรายละเอียดเฉพาะงานทั่วไป ไม่ได้แสดงรหัสที่จำเป็นต้องนำมาใช้ร่วมกันในงานเฉพาะด้าน จึงแนะนำให้หาซื้อหนังสือ Excel VBA เก็บไว้หลายๆ เล่ม หรือใช้ Google ค้นหาคำตอบยามที่ต้องการหรือติดปัญหา

ควรเลือกซื้อหนังสือที่มีสารบัญ และดัชนีเรื่องท้ายเล่ม อย่างละเอียดเพื่อสะดวกในการค้นหา หนังสือบางเล่มจะขายพร้อมกับแผ่น CD เก็บรหัส VBA ที่ใช้ในหนังสือ ช่วยให้เราสามารถลอกรหัสมาใช้ได้เลยโดยไม่ต้องเสียเวลาเขียนเอง

ที่แนะนำให้ซื้อหนังสือไว้หลายๆ เล่มนี้ ไม่ได้ให้ซื้อมาเพื่อมาเสียเวลาเปิดอ่านทุกหน้า ไม่ต้องการให้จดให้จำ แต่ให้มีหนังสือเก็บไว้ใช้ค้นหาคำตอบที่ต้องการ

หากมีข้อสงสัยอยากถามปัญหา สำหรับเว็บ Excel ภาษาไทย ไปที่

www.XLSiam.com หรือ www.ExcelExpertTraining.com สำหรับภาษาอังกฤษ ไปที่ www.MrExcel.com

เริ่มทดลอง อย่างรวดเร็ว

คนจะเก่ง Excel หรือเก่ง VBA ได้ ต้องลองฝึกลองถูกมาเยอะ อย่ามัวกลัวๆกลัวๆให้เสียเวลา ขอให้ทดลองใช้ Macro Recorder ทดลองแก้ไขรหัส ทดลองเขียนรหัสด้วยตนเอง ขอให้ถือว่า ฝึกเป็นครู เมื่อทำผิดไปแล้ว จะรู้ทำอย่างนั้นอย่างนี้ไม่ได้ ต่อไปย่อมไม่มีทางทำผิดเช่นนั้นอีก

พอเริ่มรู้จักว่าต้องระวังอะไรบ้าง เริ่มรู้จักโครงสร้างพื้นฐานของชุดรหัสคำสั่ง พอใช้งานได้โดยไม่ลำบากแล้ว จากนั้นให้ทดลองลอกรหัสของคนอื่นมาใช้ดูบ้าง ให้เปรียบเทียบกับรหัสที่คุณเขียนขึ้นมาเอง วิเคราะห์ข้อดีข้อเสีย เรียนรู้ว่ารหัสแบบใดเหมาะกับการใช้งานแบบใด

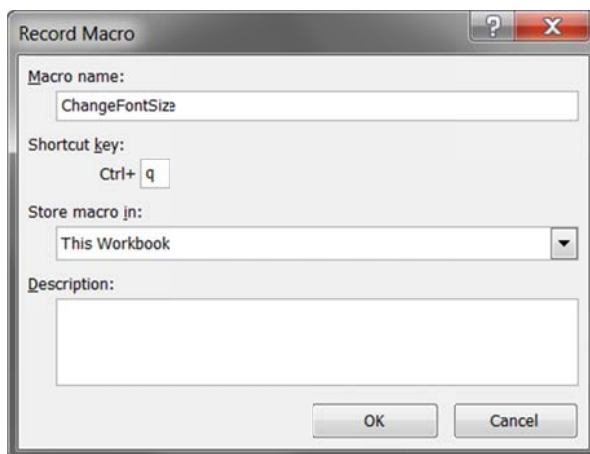
ขอให้ลอกเก็บชุดรหัสที่ทดลองว่าใช้งานได้แล้วเก็บไว้ใช้ ในอนาคตจะได้ไม่ต้องเสียเวลาพิมพ์เขียนรหัสมาใช้เองทีละตัว เพราะถ้าทุกครั้งต้องย้อนกลับมาเริ่มต้นพิมพ์

เอง ก็ต้องเสี่ยงทุกครั้งอีกนั่นแหละว่า รหัสที่เพิ่งเขียนเหล่านั้นสามารถใช้งานได้
ถูกต้องหรือไม่

ขั้นตอนสร้างรหัส VBA โดยใช้ Macro Recorder

สมมติว่า คุณต้องการจัดขนาดตัวอักษรให้กับชื่อบริษัทที่บันทึกลงในเซลล์ A1 เพื่อใช้พิมพ์เป็นหัวกระดาษ ซึ่งถูกบังคับว่า ต้องใช้ตัวอักษรขนาด 20 pixel เป็นมาตรฐานเสมอ เพื่อช่วยให้เวลาดูบนหน้าจอ หรือเมื่อพิมพ์ลงกระดาษแล้ว จะเห็นชื่อบริษัทมีขนาดเดียวกันไปตลอด

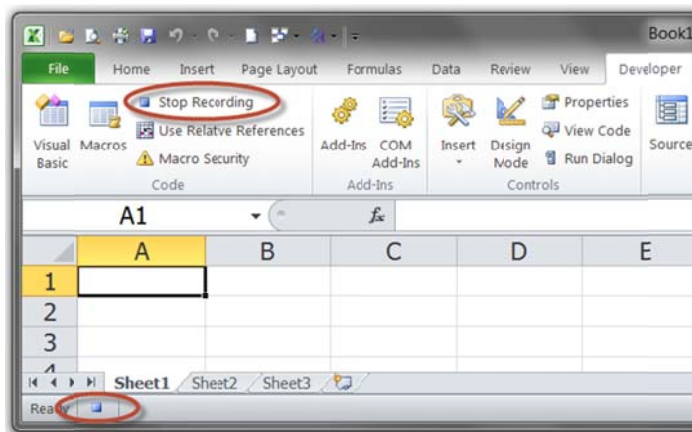
คุณอาจเลือกใช้ Macro Recorder หรือจะเลือกเขียนรหัส VBA เองก็ได้ แต่ในช่วงเริ่มเรียนรู้นี้ แนะนำให้ใช้ Macro Recorder จะช่วยทำให้งานนี้ง่ายขึ้น เพียงเริ่มจากคลิก เซลล์ชื่อบริษัท แล้วสั่ง **Developer > Record Macro**



ในช่อง **Macro name** ให้พิมพ์ชื่อซึ่งสื่อถึงงานที่ต้องการลงไป เช่น คราวนี้ต้องการบันทึกการเปลี่ยนขนาดตัวอักษร จึงใช้ชื่อ Macro ว่า **ChangeFontSize** (ชื่อ Macro ให้ใช้ตัวอักษรตัวใหญ่ผสมตัวเล็ก ไม่เว้นช่องว่างระหว่างตัวอักษร ถ้าต้องการใช้ตัวเลข ให้ใส่ตัวเลขต่อท้ายตัวอักษร)

ในช่อง **Shortcut key** ให้พิมพ์ตัว **q** ลงไป เพื่อกำหนดว่า เมื่อกดปุ่ม **Ctrl+q** เป็นการสั่งให้ Macro ทำงานซ้ำ ซึ่งในตัวอย่างนี้เป็นการสั่งให้เปลี่ยนขนาดเป็น 20 pixel (จะใช้ตัวอักษรตัวใดเป็น Shortcut key ก็ได้ แต่ไม่ควรใช้ตัว c, x, v ซึ่งซ้ำกับคำสั่งในการ Copy, Cut, หรือ Paste เพราะ Shortcut key ที่กำหนดลงไปจะทำงานแทนหน้าที่เดิม)

จากนั้นกดปุ่ม **OK** ซึ่งจะพบว่า เมนู Record Macro เปลี่ยนค่าแสดงเป็นปุ่มคำสั่ง **Stop Recording** รอไว้และด้านล่างซ้ายของจอจะมีปุ่มสี่เหลี่ยมเตือนให้ทราบว่า ขณะนี้กำลังบันทึก Macro อยู่ และสามารถคลิกที่ปุ่มนี้เพื่อสั่งหยุดการบันทึก Macro

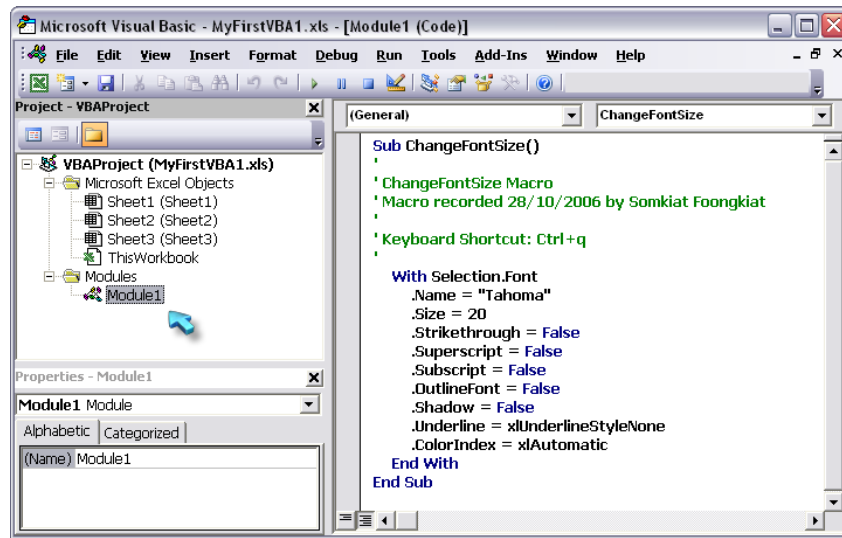


พอถึงขั้นระหว่างการบันทึก Macro นี้ต้องทำขั้นตอนต่อไปอย่างระมัดระวัง เพราะ Excel จะบันทึกและสร้างรหัส VBA ตามที่คุณใช้เมาส์หรือแป้นพิมพ์ ซึ่งถ้าคลิกผิดพลาดหรือพิมพ์อะไรผิด จะส่งผลให้เกิดรหัส VBA ทำงานตามที่ผิดนั้นด้วย

ในขั้นนี้ให้คลิกที่ช่องปรับขนาดตัวอักษรแล้วเปลี่ยนขนาดเป็น 20 จากนั้นให้กดปุ่ม **Stop Recording** แล้วจะเห็นว่าเมนูนี้ปิดไป พร้อมกับไม่มีปุ่มสี่เหลี่ยมที่เคยแสดงด้านล่างซ้ายของจออีกต่อไป แสดงว่าได้ทำการบันทึกเสร็จเรียบร้อยแล้ว

จากนั้นเมื่อต้องการปรับขนาดตัวอักษรที่เซลล์ใด ให้คลิกที่เซลล์นั้น ตามด้วยกดปุ่ม **Ctrl+q** จะพบว่า ขนาดตัวอักษรถูกเปลี่ยนเป็น 20 pixel ให้ทันที

รหัส VBA ที่เกิดขึ้นจะเก็บอยู่ภายในตัวแฟ้มนั่นเอง เพียงแต่ยังซ่อนไว้ จนกว่าจะกดปุ่ม **ALT+F11** หรือสั่ง **Developer > Visual Basic** เพื่อเปิด VBE ขึ้นมา จะพบว่า รหัสถูกบันทึกไว้ในชื่อ Module1



รหัส VBA ที่ได้จากการใช้ Macro Recorder เรียกได้ว่าเป็นรหัส Macro และมักพบว่า รหัส Macro มีโครงสร้างยาวกว่าที่จำเป็นอยู่เสมอ ซึ่งในตัวอย่างนี้ต้องการบันทึกเพียง ปรับขนาดตัวอักษร แต่ Macro Recorder จะบันทึกค่า Default เดิมทั้งหมดตามไปด้วย ถ้าคุณแก้ไขรหัส VBA เป็นจะสามารถแก้ไขรหัสให้สั้นลง จากโครงสร้างชุดคำสั่งเดิม

Sub ChangeFontSize()

```
'
' ChangeFontSize Macro
' Macro recorded 14/2/2012 by Somkiat Foongkiat
'
' Keyboard Shortcut: Ctrl+q
'

With Selection.Font
    .Name = "Tahoma"
    .Size = 20
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
End With
End Sub
```

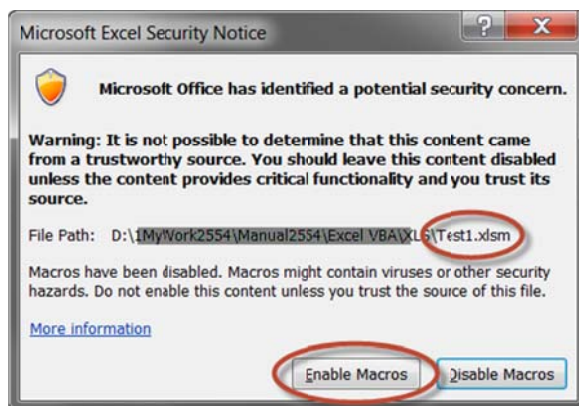
End With
End Sub

ชุดรหัสใหม่ซึ่งแก้ไขให้เหลือเท่าที่จำเป็น

Sub ChangeFontSize()
 Selection.Font.Size = 20
End Sub

ทุกคนที่มีโปรแกรม Microsoft Office จะสามารถเรียกใช้โปรแกรม Visual Basic Editor (VBE) ได้โดยไม่จำเป็นต้องซื้อเพิ่มเติม เราใช้ VBE สำหรับเปิดดูรหัสหรือเขียนแก้ไขรหัส หรือถ้าใช้ Macro Recorder เป็นเครื่องมือบันทึกขั้นตอนการทำงาน ตัว Recorder จะช่วยสร้างรหัส VBA ลงใน VBE ให้เองโดยที่เราไม่ต้องสนใจว่า VBE นี้หน้าตาเป็นอย่างไรก็ได้

สิ่งสำคัญก่อนที่จะเปิดแฟ้มที่มี Macro หรือ VBA เราจะต้องสั่งจัดเก็บแฟ้มที่มีรหัส VBA โดยกำหนดให้ใช้นามสกุล xlsx หรือ xls และเมื่อเปิดแฟ้มขึ้นมา ต้องกดปุ่ม Enable เพื่อยอมให้ Macro ทำงาน



หากไม่มีคำเตือนดังกล่าว ให้ตรวจสอบว่าคำสั่ง **File > Excel Options > Trust Center > Trust Center Settings > Macro Settings > กาช่อง Disable all macro with notifications** นี้ไว้เป็น default อยู่ก่อนแล้วหรือไม่

โครงสร้างชุดคำสั่ง VBA

ชุดคำสั่งหรือชุดรหัส VBA มาจากคำว่า Procedure ซึ่งชุดคำสั่งแต่ละชุดประกอบด้วย คำสั่งหลายบรรทัด แต่ละบรรทัดมีรหัส VBA เพื่อสั่งให้ Excel ทำงานทีละขั้น โดยทั่วไปเราควรแยกให้ชุดคำสั่งหนึ่งๆ ให้ใช้ควบคุมการทำงานที่เกี่ยวข้องกันชัดเจน

ชุดคำสั่ง VBA มี 2 ประเภท คือ

1. **Sub Procedure** เป็นชุดคำสั่งทำหน้าที่ควบคุมการทำงานทั่วไปของ Excel
2. **Function Procedure** เป็นชุดคำสั่งทำหน้าที่คำนวณคืนค่าผลลัพธ์ ใช้สำหรับสร้างสูตรใหม่มาใช้กับงานเฉพาะด้าน นอกเหนือจากสูตรสำเร็จรูปที่ Excel จัดเตรียมไว้ให้

ลักษณะโดยทั่วไปของชุดคำสั่งแต่ละชุด ถ้าเป็น Sub Procedure จะอยู่ในช่วงรหัส ตั้งแต่ Sub จนถึงคำว่า End Sub ส่วน Function Procedure ชุดหนึ่งๆ จะอยู่ในช่วงรหัสตั้งแต่ Function จนถึงคำว่า End Function (ซึ่งชุดคำสั่งที่ Macro Recorder สร้างขึ้น จะเป็นชุดคำสั่งแบบ Sub Procedure เท่านั้น)

หมายเหตุ ยังมีชุดคำสั่งประเภท Class Procedure ซึ่งยากเกินกว่าจะนำมาอธิบายในที่นี้

โครงสร้างชุดคำสั่งแบบ Sub Procedure

Sub ชื่อชุดคำสั่ง()

'comment

รหัสคำสั่งแต่ละบรรทัด

รหัสคำสั่งแต่ละบรรทัด

รหัสคำสั่งแต่ละบรรทัด

End Sub

ลองเปรียบเทียบชุดคำสั่งที่ได้จาก Macro Recorder ที่ใช้เปลี่ยนขนาดตัวอักษรเป็น 20 pixel ดังนี้

```
Sub ChangeFontSize()
'
' ChangeFontSize Macro
' Macro recorded 14/2/2012 by Somkiat Foongkiat
'
' Keyboard Shortcut: Ctrl+q
'

With Selection.Font
    .Name = "Tahoma"
    .Size = 20
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
End With
End Sub
```

ส่วนที่ระบุว่าเป็นชุดคำสั่ง

```
Sub ChangeFontSize()
    xxxxxxxxxxxxxxxx
End Sub
```

คำว่า ChangeFontSize เป็นชื่อของชุดคำสั่งนี้ ต้องตามด้วยเครื่องหมาย () เสมอ

ส่วนที่เป็น Comment

```
'
' ChangeFontSize Macro
```

```
' Macro recorded 14/2/2012 by Somkiat Foongkiat
```

```
'
```

```
' Keyboard Shortcut: Ctrl+q
```

```
'
```

Comment เป็นบรรทัดที่ไม่ได้เกี่ยวข้องกับการทำงาน ใช้สำหรับบันทึกคำอธิบาย
สื่อสารกับมนุษย์ มักใช้เขียนคำอธิบายหน้าที่ของรหัส เพียงพิมพ์เครื่องหมายฝนทอง '
นำหน้า แล้วตามด้วยคำอธิบายที่ต้องการ

แทนที่จะลบรหัสทิ้ง เราสามารถสั่งให้รหัสบรรทัดนั้นไม่ทำงาน โดยพิมพ์เครื่องหมาย
ฝนทองนำหน้าบรรทัดนั้น หรือถ้าต้องการสั่งหยุดรหัสบางส่วนในบรรทัดให้หยุดทำงาน
ให้พิมพ์เครื่องหมายฝนทอง นำหน้าส่วนของรหัสที่ต้องการหยุดทำงาน โดย VBA จะ
ถือว่าเมื่อพบเครื่องหมายฝนทอง จะแสดงว่า ตั้งแต่เครื่องหมายฝนทองนั้นไปจนสุด
บรรทัด ถือเป็น Comment ไม่ต้องนำมาเกี่ยวข้องกับการทำงาน

ส่วนของรหัส VBA

```
With Selection.Font
    .Name = "Tahoma"
    .Size = 20
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
End With
```

สังเกตว่า ด้านซ้ายของรหัสจะถูกย่อหน้าไว้ เพื่อช่วยให้มนุษย์อ่านง่ายขึ้น และมี
เครื่องหมายจุด . แทรกอยู่แทบทุกบรรทัด โดย VBA จะใช้เครื่องหมายจุดนี้ แบ่งรหัส
ออกเป็นส่วนใหญ่ตามด้วยส่วนย่อย จากซ้ายไปขวา เช่น Selection.Font มีความหมาย
ว่า ในส่วนที่เลือกนั้น.ตัวอักษร

นอกจากนั้นสาเหตุที่มีคำว่า With (ซึ่งจบด้วยคำว่า End With) เป็นการช่วยให้รหัสลดความซ้ำซ้อนลง ทำให้ไม่จำเป็นต้องเขียนรหัสซ้ำๆกัน ในช่วงท้ายมือของรหัสทุกบรรทัด ดังนี้

Selection.Font.Name = "Tahoma"

Selection.Font.Size = 20

.....

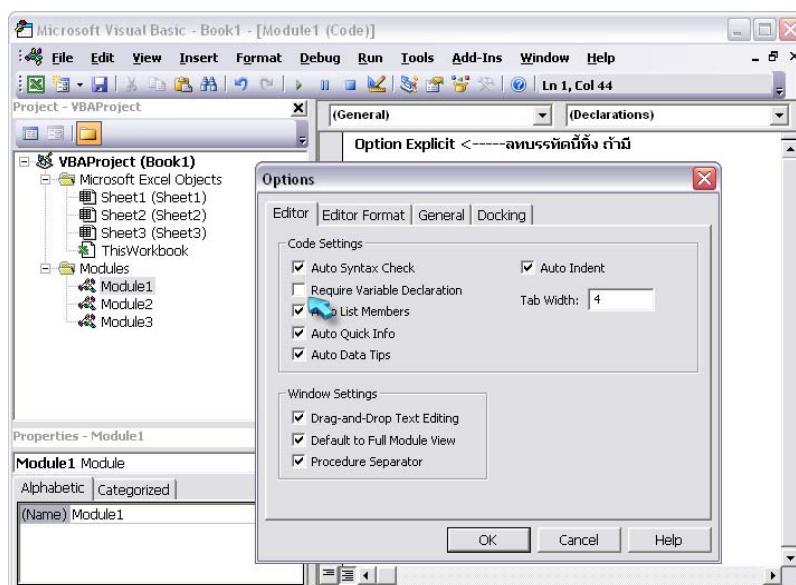
จะเห็นได้ว่า โครงสร้างรหัส VBA นั้น ใกล้เคียงกับภาษาที่คนเราใช้สื่อสารกันนั่นเอง แทนที่ต้องพูดสั้นๆซ้ำๆว่า

ในส่วนที่เลือกนั้น.ตัวอักษร.ชื่อตัวอักษร ทำให้เป็น Tahoma

ในส่วนที่เลือกนั้น.ตัวอักษร.ขนาดตัวอักษร ทำให้เป็น 20

สามารถสังเกตสั้นลงว่า ในส่วนที่เลือกนั้น.ตัวอักษร นั้น ให้ใช้ตัวอักษรใด ขนาดใด ซึ่งง่ายต่อการเขียน ประหยัดเวลา และยังทำให้รหัสทำงานเร็วขึ้นด้วย

ข้อควรสังเกตและข้อควรระวังในการเขียนรหัส VBA



1. เมื่อกดปุ่ม **ALT+F11** เพื่อเปิด VBE ขึ้นมาแล้ว ไม่มีจอ Window ภายใตแบ่งเป็นส่วนๆ ให้ใช้เมนูของ VBE สั่ง **View > Project Explorer** และ

View > Properties Window เพื่อแบ่งจอด้านซ้ายให้แสดงชื่อแฟ้ม และโครงสร้างของแฟ้ม ปรากฏให้เห็นใน Project Explorer (ส่วน Properties Window นั้น ไม่จำเป็นต้องเปิดขึ้นก็ได้)

2. ให้สั่งปรับระบบภาษาที่ใช้ให้เป็นภาษาไทย ในเมนูของ VBE ให้สั่ง **Tools > Options > Editor Format > Font** เลือก Font ไทย (แนะนำ Tahoma Bold(Thai))
3. ชุดคำสั่งที่ใช้งานทั่วไป ให้เขียนเก็บไว้ในชีทชื่อ Module1, Module2, Module3 ...โดยใช้เมนู **Insert > Module** ใน VBE จะช่วยสร้าง Module โดยเรียงเลขต่อท้ายให้เรื่อยไป
4. ให้สังเกตว่า ในหน้าจอส่วนของ Project Explorer เมื่อดับเบิลคลิกที่ชื่อ Module1 จะมีสีพื้นใต้ชื่อเป็นสีเทาจาง พร้อมกันนั้นจอด้านขวาจะเปิด Code Window เป็นพื้นที่ใส่เก็บรหัสเปิดขึ้นมาให้เห็น และด้านบนซ้ายของจอ จะแสดงชื่อแฟ้มต่อด้วยชื่อ Module เช่น Book1.xlsx [Module1(Code)] ทั้งนี้เพื่อแสดงให้เห็นชัดว่า Code Window ที่กำลังใช้งานนั้นเป็นของ Module1 ชัดเจน (อย่าดูแค่สีใต้ชื่อ Module1 เพราะถ้าลองคลิกที่ชื่อ Module1 จะเปลี่ยนสีพื้นเป็นสีเข้ม ซึ่งต้องดับเบิลคลิกจึงจะเป็นสีเทาจาง)
5. ในขั้นตอนที่เพิ่งเริ่มเรียนรู้ VBA นั้น ให้ตรวจสอบ VBE ว่า **Tools > Options > Editor ไม่ได้กาช่อง Require Variable Declaration** (หากพบว่าเดิมกาไว้ ให้ตัดกาทิ้งไป ซึ่งคำสั่งนี้ถ้ากาไว้ จะทำให้ทุกครั้งที่ Insert > Module ใหม่ จะมีคำสั่ง Option Explicit เขียนไว้ในบรรทัดแรก ดังนั้นถ้าพบคำว่า Option Explicit ก็ให้ลบทิ้งบรรทัดนี้ออกไปด้วย)
6. ควรแยกเก็บชุดคำสั่งที่ทำงานต่างกันโดยสิ้นเชิง ไว้ต่าง Module กัน
7. หากใช้ Macro Recorder ช่วยสร้างรหัส ควรบันทึกขั้นตอนสั้นๆแยกออกจากกันในแต่ละครั้ง เพื่อทำให้เกิด Sub Procedure แยกเก็บรหัสแต่ละชุดออกจากกัน ซึ่งทุกครั้งที่เปิดแฟ้มขึ้นมาบันทึก Macro ใหม่ จะพบว่า VBE เปิด Module ใหม่แยกจาก Module เดิมเสมอ
8. ถ้าต้องการกำหนด Shortcut Key ให้กับชุดคำสั่งที่เขียนเอง หรือ เปลี่ยนแปลง Shortcut Key ชุดคำสั่งที่ใช้ Macro Recorder ให้กลับที่

Excel แล้วสั่ง **Developer > Macros >** คลิกเลือกชื่อชุดคำสั่ง แล้วกดปุ่ม Options... จะเห็นช่องที่พิมพ์ Shortcut Key ลงไปได้ (สังเกตว่า Shortcut Key ที่เขียนไว้ในส่วนของ Comment อาจไม่ตรงกับ Shortcut Key ที่แก้ไขใหม่ก็ได้ เพราะ VBE ไม่ได้ตามแก้สิ่งที่บันทึกเดิมไปแล้ว)

9. ให้พิมพ์ตัวอักษรตัวเล็กเสมอ เว้นเฉพาะชื่อ Sub หรือชื่อตัวแปรที่คุณตั้งเองให้พิมพ์ตัวเล็กผสมตัวใหญ่ตามต้องการ เมื่อกด Enter หรือขึ้นบรรทัดใหม่จะพบว่า VBE ช่วยแก้ไขตัวอักษรตัวเล็กที่คุณพิมพ์ หากเป็นคำสั่งที่ VBE รู้จัก จะถูกแก้ไขเป็นตัวอักษรตัวใหญ่ผสมตัวเล็กต่อให้เอง แต่ถ้าเป็นคำสั่งที่สะกดผิดหรือ VBE ไม่รู้จัก ให้สังเกตว่ายังคงเป็นตัวเล็กตามเดิม
10. เมื่อพิมพ์คำว่า Sub ตามด้วยชื่อของ Sub นั้นแล้วกดปุ่ม Enter จะพบว่า VBE จะพิมพ์เครื่องหมาย () ต่อท้ายชื่อ Sub ให้เอง และเกิดคำว่า End Sub พิมพ์ในบรรทัดต่อไปเพื่อแสดงให้เห็นจุดสิ้นสุดของ Sub
11. VBE จะย้ายไปแก้ไขชื่อตัวแปรที่สะกดตรงกันให้เป็นตัวใหญ่ตัวเล็ก ตามชื่อตัวแปรตัวล่าสุดที่พิมพ์ลงไปให้เสมอ เช่น เดิมใช้ชื่อตัวแปร myVar แต่ต่อมาคุณพิมพ์ MYVar จะพบว่า VBE แก้ไข myVar เดิมทุกตัวให้เป็น MYVar
12. ควรจัดย่อหน้ารหัส VBA ในชุดคำสั่งให้ย่อหน้าลดหลั่นกันไป เพื่อใช้แนวย่อหน้าที่ตรงกัน แสดงถึงคำสั่งระดับเดียวกัน
13. ถ้ารหัสแต่ละบรรทัดยาวมาก สามารถจัดรหัสขึ้นบรรทัดใหม่ได้ โดยเคาะวรรคแล้วพิมพ์เครื่องหมาย _ แล้วกดปุ่ม Enter เพื่อพิมพ์ต่อในบรรทัดใหม่
14. ใน VBE จะจดจำทุกขั้นตอนเดิมที่คุณทำไว้ตลอด ช่วยให้สามารถ Undo หรือกดปุ่ม Ctrl+z เพื่อย้อนกลับไปใช้รหัสเดิมที่เขียนไว้ได้ทุกขั้นที่ผ่านไปแล้ว
15. ควรสั่ง Save ก่อนที่จะสั่ง Run รหัสเสมอ เพราะหากรหัสไม่ทำงาน ในบางครั้งอาจทำให้เครื่อง Hang และไม่สามารถสั่ง Save เก็บรหัสที่อุตสาหะเสียแรงเสียเวลาสร้างเก็บไว้ได้
16. สั่งรหัสให้ทำงาน โดยคลิกในพื้นที่ระหว่าง Sub ถึง End Sub ของชุดคำสั่งที่ต้องการ แล้วกดปุ่ม **F5** หรือ ใช้เมนู **Run > Run Sub** หรือ

คลิกที่ปุ่ม Run (เป็นรูปลูกศรขวา) หากต้องการให้รหัสทำงานที่ละบรรทัด ให้กดปุ่ม **F8** ต่อกันไปเรื่อยๆ

17. ถ้าต้องการสั่งรหัสให้ทำงานโดยสั่งจาก Excel ให้ใช้คำสั่งบนเมนู

Developer > Macros จากนั้นคลิกเลือกชื่อชุดคำสั่งที่ต้องการ แล้วกดปุ่ม Run เพื่อสั่งให้ทำงานต่อเนื่องกันทั้งหมด หรือกดปุ่ม Step Into เพื่อสั่งให้ทำงานที่ละบรรทัด (ซึ่งต้องกดปุ่ม F8 ต่อไปเอง)

18. ในกรณีรหัสหยุดทำงานทั้งๆที่รหัสยังทำงานไม่ครบทั้งหมดคำสั่ง จะเห็นว่า VBE เปลี่ยนสีพื้นบรรทัดของรหัสที่ไม่ทำงานเป็นพื้นสีเหลือง และจะทำให้ทั้ง VBE และ Excel หยุดค้างตามไปด้วย ให้สั่ง **Run > Reset** หรือคลิกที่ปุ่ม Reset (เป็นรูปสี่เหลี่ยมจัตุรัส อยู่ติดกับปุ่ม Run) ก่อน พื้นสีเหลืองจะหายไป จากนั้นจึงจะสามารถกลับไปตรวจสอบแก้ไขรหัสต่อไป

19. เมื่อทดสอบรหัสทั้งหมดว่าทำงานถูกต้องสมบูรณ์แล้ว ควร Export

Module ออกไปแล้ว Remove Module ตามไปด้วย จากนั้นสั่ง Save แล้วเปิดแฟ้มขึ้นมาใหม่ แล้วจึง Import Module กลับเข้ามาเหมือนเดิม ทั้งนี้เพื่อช่วยลบขยะที่ตัว Module เก็บไว้ตลอดช่วงที่คุณแก้ไขรหัสทั้งออกไป ทำให้แฟ้มมีขนาดเล็กลง

ขั้นตอนนี้ให้คลิกขวาที่ชื่อ Module แล้วสั่ง Export File ออกไปเป็นแฟ้มใหม่มีนามสกุล bas แล้วคลิกขวา Remove Module จากนั้นคลิกขวาที่ตัว Module สั่ง Import File .bas เข้ามาใหม่

วิธีใช้ VBA ให้คุ้มค่า

หลายคนพอเริ่มใช้ Macro Recorder เป็น จะเห็นว่าวิธีสร้างรหัส VBA ขึ้นมาใช้งาน ไม่ได้ยากอย่างที่คิด ที่แท้แค่สั่ง Record Macro เท่านั้น จากนั้นค่อยใช้รหัส VBA ทำงานอย่างไร ขอให้ใช้เมาส์และแป้นพิมพ์ทำงานตามแบบที่ต้องการ จะเกิดรหัส VBA ขึ้นตามต่อมาให้โดยเราไม่ต้องเขียน VBA เองแม้แต่หน่อย

จากขั้นตอนง่ายๆจากการใช้ Macro Recorder นี้แหละ กลายเป็นตัวกระตุ้นให้หาทางใช้ VBA ต่อกัน บางคนพอสามารถใช้ VBA ควบคุมแฟ้ม Excel ให้ทำงานได้ดังใจ ก็จะยกแฟ้มที่สร้างด้วยฝีมือตนเองให้คนอื่นดูด้วยความภาคภูมิใจ โดยหารู้ไม่ว่า ผลงานที่ตนชื่นชมอย่างยืงนั้น ไม่ได้คุ้มค่าอย่างที่คิด

คุ้มค่า ค่าของอะไร เทียบกับอะไร

ลองนึกดูซิว่า ถ้าคุณเสียเวลาสร้างรหัส VBA นานนับชั่วโมง แต่รหัส VBA ที่ใช้นั้น สามารถช่วยประหยัดเวลาการทำงานได้แค่ไม่กี่นาที ในกรณีเช่นนี้จะถือว่าคุ้มค่าไหม แต่ต่อมาถ้าต้องใช้รหัสนั้นซ้ำแล้วซ้ำอีก เมื่อนับรวมเวลาที่ประหยัดได้แล้วเกินกว่าเวลาที่ใช้สร้าง VBA อย่างนี้จึงเรียกได้ว่า คุ้มใช่ไหม.... ยังหกรอก คิดเทียบแค่นี้ง่ายเกินไป

กว่าจะใช้ VBA เป็น คุณต้องลงทุนลงแรงอะไรบ้าง

1. เสียเวลาเข้ามาอบรม
2. เสียเงินซื้อหนังสือ VBA มาอ่าน
3. เสียเวลาลองผิดลองถูกตอนฝึกใช้ VBA
4. เสียเวลาเขียนคู่มือ บันทึกขั้นตอนและวิธีการที่ใช้
5. เสียเวลาประกาศตัวแปร เพื่อให้รหัสทำงานได้เร็วขึ้น
6. เสียเวลาปรับปรุงดัดรหัส VBA ที่ไม่จำเป็น ทำให้สั่นกะทัดรัด
7. เสียเวลาย้อนกลับมาแกะรหัสเดิมที่เขียนไว้นานแล้ว เพื่อใช้ใหม่
8. เสียโอกาสใช้เวลาให้กับชีวิตส่วนตัว เสียความสุขในชีวิตไปบางส่วน

จากนี้ขอยกเฉพาะประเด็นที่ควรแก้การทำความเข้าใจหรืออาจเข้าใจไม่ตรงกัน มาอธิบาย

เสียเวลาเข้าอบรม

หลักสูตรอบรมที่ว่าด้วยวิธีใช้ VBA ถ้าจะให้สมบูรณ์ จะต้องใช้เวลาอบรมนานเป็นเดือนเรียนกัน ให้การบ้านทำมาส่งอาจารย์กัน เหมือนกับสมัยที่เราเรียนวิชาหนึ่งๆ สมัยมหาวิทยาลัยนั่นแหละ จึงจะได้ผลสมบูรณ์ แต่ถ้าให้เวลาอบรมเพียงไม่กี่ชั่วโมง เรียนติดต่อกันเพียงไม่กี่วัน ถือว่ายากมากที่จะเกิดผลตามต้องการ ใหนจะต้องแบ่งเวลาให้กับภาคทฤษฎี ตามด้วยเวลาที่เหลือ มาใช้ปฏิบัติสร้างงานกัน

ถ้าเรียนติดต่อกัน 3 วัน แล้วใช้เวลาให้กับการทดลองใช้ VBA แคตัวอย่างเดียวหรือไม่ก็ตัวอย่าง พอจบอบรมออกมา แทบทุกคนจะนำความรู้ที่เพิ่งอบรมจบไป นำมาใช้ได้กับงานที่ตรงกับตัวอย่างที่เรียนมาเท่านั้น ส่วนงานที่ไม่ตรงกับตัวอย่างที่อบรม ต้องปล่อยไว้อย่างเดิมนั่นแหละ เพราะคิดต่อเองไม่เป็น บางคนพออบรมเสร็จ ก็คืนความรู้ที่เรียนไปคืนให้อาจารย์ทันที เพราะตัวอย่างที่เรียนไปนั้น ไม่เห็นตรงกับงาน ไม่ตรงกับปัญหาของตนที่ต้องการเอาความรู้ที่เรียนไปกลับไปแก้ปัญหา

ที่แย่กว่านั้น ถ้าติดปัญหาสงสัยในระหว่างอบรม แล้วไม่กล้าถาม ไม่กล้ายกมือบอกว่าไม่เข้าใจ ขอให้สอนซ้ำ ยิ่งเรียนต่อนานเข้า ปัญหาค้างคาใจจะมีมากขึ้นจนกลายเป็นดินพอกหางหมู สมองของผู้เข้าอบรมคนนั้นมักจะปิดรับรู้อะไรที่อบรมส่วนที่เหลือ ที่ว่าอบรมกัน 3 วัน จึงกลายเป็นว่า รู้เรื่อง ตามเนื้อหาทันแต่วันแรกเท่านั้น ส่วนความรู้ที่เหลืออีก 2 วัน หายสาบสูญไปในอากาศ

ดังนั้นถ้าต้องเสียเวลาทำงานมาเข้าอบรม แล้วอยากได้ผลคุ้มค่ากับเวลาที่เสียไป ต้องหาทางใช้เวลาระหว่างการอบรมให้ยืดหยุ่นที่สุด มุ่งเน้นให้คิดเองทำเองต่อให้เป็น อย่าเอาแต่ท่องจำ ขอให้ใช้เวลาทำตัวอย่างหลายๆแบบ ปฏิบัติมากๆ ส่วนทฤษฎีตามมาก็หลัง หัวข้อเนื้อหาหลักสูตรควรมุ่งที่ตัวปัญหาในงานว่า จะใช้ VBA อย่างไรกลับมาใช้กับปัญหานั้น อย่าเรียนแบบไล่ตามเนื้อหาของ VBA ว่ามีอะไรบ้าง

คุณควรถือว่า เวลาที่เสียไปกับการอบรมเพียง 3 วัน เป็นเพียงจุดเริ่มต้นที่ช่วยให้คุณเองได้เรียนรู้ เป็นแนวทางให้คุณสามารถย้อนกลับมาลองฝึกทดลองดู และค่อยๆเพิ่มพูนความรู้เรื่อง VBA ให้ลึกขึ้นเองในภายหลัง ซึ่งต้องใช้เวลาศึกษาเองต่ออีกนานนับปีทีเดียว ถ้าคุณตั้งหลักคิดได้แบบนี้ รับรองว่าที่เสียเวลามาอบรม VBA จะคุ้มค่าทีเดียว

เสียเงินซื้อหนังสือ VBA มาอ่าน

เรื่องหนังสือ VBA นี้ต้องช่วยทำใจกันหน่อย เพราะจะหาหนังสือ VBA ที่ให้คำตอบที่เราต้องการไม่ค่อยได้หรอก แม้หนังสือ VBA โดยทั่วไปเป็นหนังสือที่ดี แต่ที่ว่า ดี ก็เพราะมีเนื้อหาให้รายละเอียดได้ดีมาก ไล่เรียงจากที่ไปทีมา ว่าด้วย VBA คืออะไร มีหน้าตาของรหัสเป็นอย่างไร ต้องเขียนโครงสร้างรหัสอย่างไร มีกฎเกณฑ์อะไรที่ต้องระวัง จากนั้นมักเป็นเนื้อหาชั้นยาก ยากแบบที่เรียกว่า กระโดดข้ามชั้นไปยากสุดๆกันเลย และตัวอย่างที่ไขก็ไม่ค่อยตรงกับที่เราต้องการ

หนังสือ VBA ที่คุณควรหาซื้อมาเก็บไว้ ไม่ควรมีเนื้อหาแค่ VBA แต่ควรมีเนื้อหาอธิบายครอบคลุมถึงความสัมพันธ์ของ VBA กับ Excel ด้วย โดยขอแนะนำหนังสือของ John Walkenbach ซึ่งมีเนื้อหาที่อ่านง่าย ไล่เรียงตั้งแต่ Excel เรื่อยไปจนถึงตัว VBA และมีตัวอย่างประกอบไปตลอด ต่างจากหนังสือของผู้แต่งคนอื่นที่ขอบเริ่มบทแรกเป็น VBA และใช้ตัวอย่างยากๆ ที่คนเก่งๆเห็นแล้วก็ยังง

อย่างไรก็ตามคุณควรหาซื้อหนังสือ VBA ไว้หลายๆเล่ม จากผู้เขียนหลายๆคน เพราะหนังสือแต่ละเล่ม มักไม่ได้รวบรวมคำตอบที่คุณต้องการทั้งหมดไว้ด้วยกัน ขอให้เก็บหนังสือไว้ใช้ค้นหาคำตอบเมื่อคุณติดปัญหา ลองนึกดูว่า ถ้าคุณใช้ VBA แล้วติดขัดขึ้นมา งานไม่เดินนานเป็นชั่วโมงหรือเป็นวัน ช่วงเวลาที่งานหยุดนิ่งนี้แหละ คิดต้นทุนที่เสียไปเป็นเงินเป็นทองได้มหาศาล แต่ถ้ามีหนังสือให้พลิกค้นหาคำตอบ สามารถแก้ไขปัญหาก็ติดค้างอยู่ได้ทันที อย่างนี้จึงเรียกว่า ซื้อหนังสือมาแล้วคุ้ม

เสียเวลาประกาศตัวแปร เพื่อให้รหัสทำงานได้เร็วขึ้น

การประกาศตัวแปร มาจากคำว่า Declare Variables เป็นคำสั่งที่กำหนดให้เขียนขึ้นในบรรทัดแรกของรหัส VBA ทั้งหมด เพื่อกำหนดให้คอมพิวเตอร์ใช้หน่วยความจำเท่าที่

อย่างไร เนื่องจากรหัสจะทำงานได้เร็วหรือช้า นั้น มิได้ขึ้นกับการประกาศตัวแปรเพียงอย่างเดียว และการประกาศตัวแปรที่ไม่เหมาะสม อาจทำให้รหัสไม่ทำงานตามที่คิดไว้ แล้วกว่าจะหาคำตอบเจอว่าต้องแก้ไขในส่วนของการประกาศตัวแปร อาจต้องเสียเวลา ค้นหาจุดผิดพลาด หาแล้วหาอีกก็ยังไม่พบเพราะคิดไม่ถึงว่า ปัญหาเกิดจากตัวแปรนี้เอง

นอกจากนี้ลองพิจารณาว่า ถ้าต้องเสียเวลาหลายนาที่คิดเขียนประกาศตัวแปร แล้วทำให้แฟ้มงานทำงานเร็วขึ้นเพียงแค่เสี้ยววินาที อย่างนี้เรียกว่า คุ่มหรือไม่ ต่อเมื่อคุณมีข้อจำกัดว่า เครื่องคอมพิวเตอร์ที่ใช้ทำงานนั้นเป็นรุ่นเก่า ทำงานช้า หรือไม่สามารเพิ่มหน่วยความจำให้มากขึ้นต่างหาก จึงควรพิจารณาหาทางใช้การประกาศตัวแปรให้เกิดประโยชน์

เสียเวลาปรับปรุงสูตรรหัส VBA ที่ไม่จำเป็น ทำให้สิ้นกะทัดรัด

ประเด็นนี้คล้ายคลึงกับการเขียนประกาศตัวแปร แทนที่จะเสียเวลาเขียนประกาศตัวแปร ซึ่งทำให้รหัสยาวขึ้น กลับมาเสียเวลาปรับปรุงรหัส สูตรรหัส VBA ที่ไม่จำเป็น เพียงแค่ทำให้ดูว่ารหัสสิ้นกะทัดรัดขึ้น โดยไม่ได้ช่วยให้รหัสทำงานเร็วขึ้นจนถึงถือว่า ขดเขย คุ่มค่ากับเวลาที่เสียไปในการปรับปรุงรหัสให้สั้นลง

อยากให้ VBA ให้คุ้มค่า ต้องคิดต้องทำอะไร

กฎเกณฑ์ต่อไปนี้เป็นเพียงความเห็นหนึ่ง อยากให้ถือเป็นเพียงคำแนะนำ ไม่ใช่กฎเกณฑ์ตายตัว หรือต้องบังคับใช้กับทุกคน ทั้งนี้ขึ้นกับสภาพแวดล้อมและสถานการณ์ของผู้ใช้ VBA ซึ่งมีความแตกต่างกันไปอีกด้วย

- อย่าคิดทำอะไรเกินตัว ตนมีความรู้ความสามารถระดับใด มีเวลาจำกัดแค่ไหน ให้ทำแค่นั้น เท่าที่ตนทำได้
- มุ่งทำงานให้เสร็จทันเวลา โดยไม่ต้องห่วงเรื่องความสมบูรณ์ของตัวรหัส ขนาดหน่วยความจำว่าสิ้นเปลืองอย่างไร หรือความเร็วของรหัส ต่อเมื่อมีเวลาเหลือพอ จึงย้อนกลับมาปรับปรุงรหัสให้สมบูรณ์

- เลือกใช้วิธีหาคำตอบที่เหมาะสมกับงานและผู้ใช้ ถ้าผู้ใช้งานรวมกัน ไม่มีความรู้เรื่อง VBA ก็ไม่ควรใช้ VBA ให้เลือกใช้วิธีอื่นที่อาจเสร็จช้ากว่า มีขั้นตอนยาวกว่า แต่ไม่มีผลเสียหรือความเสี่ยงในระยะยาว
- VBA เหมาะกับงานที่มีขั้นตอนชัดเจนแน่นอนตายตัวแล้วเท่านั้น
- เรียนรู้ VBA จากการทดลองใช้งานง่ายๆ มีรหัสสั้นๆ แต่ต้องพยายามฝึกใช้บ่อยๆ เรียนรู้จากข้อผิดพลาดของตนเอง
- ควรเลือกใช้ VBA กับงานที่ต้องนำกลับมาใช้ซ้ำแล้วซ้ำอีก หรืองานที่มีขั้นตอนยุ่งยากสลับซับซ้อน จนผู้ปฏิบัติงานอาจเผลอทำงานผิดพลาด อย่าใช้ VBA กับงานที่ใช้เพียงครั้งเดียว
- พยายามใช้ Macro Recorder ช่วยสร้างรหัส VBA ให้มากที่สุดแทนการเขียนรหัสเอง
- จัดบันทึกอธิบายการใช้ VBA ในแต่ละแฟ้มงาน แสดงรายละเอียดด้วยว่าทำไมจึงเลือกใช้วิธีนั้น ทำไมจึงไม่เลือกใช้วิธีอื่น
- ควรมีผู้ร่วมงานร่วมรับรู้การสร้าง VBA เพื่อเป็นตัวแทนตัวตายตัวแทน

ทางออกที่ดีกว่าการใช้ VBA

ข้อผิดพลาดที่ร้ายแรงที่สุดในการใช้ VBA เกิดจากการตัดสินใจใช้ VBA ทั้ๆที่มีทางออกอื่นที่ดีกว่า ง่ายกว่า สะดวกกว่าอยู่แล้ว ทำให้ต้องเสียแรงเสียเวลาสร้าง รหัส VBA ขึ้นมา พอสร้างเสร็จ ยังต้องเสียเวลาย้อนกลับมาแก้ไขอย่างยากเย็นเสียอีก

ในตัวโปรแกรม Excel มีเครื่องมือที่เป็นเมนูคำสั่งและสูตรสำเร็จรูป ซึ่งสามารถนำมาช่วยควบคุมการทำงานได้เช่นเดียวกันกับการใช้ VBA หรือนำมาใช้ร่วมกับ VBA

เครื่องมือช่วยในการเปิดแฟ้มที่ต้องเปิดพร้อมกัน

เริ่มจากจัดเปิดแฟ้มที่ต้องการทั้งหมดขึ้นมามบนจอ จากนั้นสั่ง **View > Save Workspace** ซึ่งจะเกิดแฟ้มใหม่เพิ่มอีกแฟ้มหนึ่งมีนามสกุล .xlw ต่อมาหากต้องการเปิดแฟ้มชุดเดิมทั้งหมดพร้อมกันอีก ให้สั่งเปิดเฉพาะแฟ้ม .xlw นั้นแฟ้มเดียว Excel จะไล่เปิดแฟ้มทุกแฟ้มที่ต้องการต่อให้เอง

แฟ้ม .xlw เป็นแฟ้มขนาดเล็กมาก ทำหน้าที่เก็บแค่ชื่อแฟ้มที่เคยเปิดขึ้นมาทำงานพร้อมกัน แฟ้มนี้ไม่ได้เก็บตัวแฟ้มอื่นไว้แบบ zip ดังนั้นถ้าต้องการส่งแฟ้มให้ผู้อื่นนำไปใช้งานต่อ ต้องส่งแฟ้ม .xlw นี้ไปพร้อมกับแฟ้มอื่นๆที่ต้องการเปิดพร้อมกันด้วยเพียงระบุว่า เมื่อต้องการเปิดแฟ้ม ให้เปิดแฟ้ม .xlw เพียงแฟ้มเดียว แล้วจะพบว่าแฟ้มอื่นๆจะถูกเปิดต่อให้เอง

ถ้าแฟ้มมีหลายชีท อยากจะคลิกง่ายๆให้ไปที่ชีทที่ต้องการ

คุณสามารถใช้ Hyperlink สร้างรูปภาพสวยๆไว้ในตารางเพื่อคลิก link ไปยังชีทที่ต้องการ โดยเริ่มจากวาดรูปใส่ลงไปตารางก่อน แล้วคลิกขวาที่รูป สั่ง **Hyperlink** แล้วเลือก **Place in this document** หรือ **Existing File or Web Page** เพื่อกำหนดตำแหน่งปลายทางในแฟ้มเดิมหรือในแฟ้มอื่นตามลำดับ

นอกจากการใช้ Hyperlink ที่สร้างจากเมนู ยังมี Hyperlink ที่สร้างจากสูตรได้อีก เช่น หากต้องการสร้าง link ในคำว่า Click here ไปที่ชีทชื่อ June เซลล์ E56 ใช้สูตรต่อไปนี้

```
=HYPERLINK("[Book1.xls]June!E56", "Click here")
```

อยากไปที่ชีทที่ต้องการ แล้วให้ปรับโครงสร้างตารางในชีทนั้นด้วย

เป็นเรื่องหน้าเบื่ออย่างมาก หากแฟ้มที่ใช้งานมีมากมายหลายชีท ชื่อชีทด้านล่างอาจจะเขียนต่อกันเป็นแถวยาวออกหน้าจอ ต้องเสียเวลาขยับไล่หาชีท จึงจะพบชีทที่ต้องการ และยิ่งอาจพบปัญหาต่อเนื่องไปอีกว่า ชีทที่อยากเจอนั้น ต้องคอยซ่อนแถว และปรับขนาดแถวใหม่จนกว่าจะได้โครงสร้างตารางตรงใจผู้ใช้ ซึ่งบางคนหาทางออกโดยสร้างชีทที่มีข้อมูลเหมือนกันหลายๆชีทขึ้นมา แล้วจัดโครงสร้างให้ต่างกันไปแล้วแต่ว่าตารางนั้นชีทนั้นจัดไว้ให้ใครดู ถ้าทำอย่างนี้แฟ้มจะใหญ่กว่าเดิมขึ้นเยอะทีเดียว

ปัญหาดังกล่าวจะหมดไปโดยเริ่มจากจัดโครงสร้างตารางให้ตรงความต้องการก่อน แล้วใช้คำสั่ง **View > Custom Views** ตั้งชื่อแบบชีทที่ต้องการตามที่เห็นบนจอ ทำเช่นนี้เข้าไปเรื่อยๆ โดยไม่จำเป็นต้องสร้างชีทซ้ำกันขึ้นอีก

เมื่อใดที่ต้องการไปที่ชีทใด ให้คลิกที่ **View > Custom Views** แล้วคลิกชื่อแบบชีทที่ตั้งชื่อไว้ จะพบว่า Excel พาคุณไปที่ชีทนั้นแล้วปรับโครงสร้างตารางให้ด้วย และเมื่อคุณสั่งพิมพ์ยังสามารถปรับหัวตาราง ท้ายตารางในหน้ากระดาษให้แตกต่างกันไปได้อีกด้วย เนื่องจาก Custom Views บันทึก Page Setup ไปพร้อมกัน

นอกจากนี้ควรใช้คำสั่ง **Data > Group** ร่วมกับ Custom View เพื่อทำให้ชีทหนึ่งๆ สามารถจัดโครงสร้างตารางให้เหมาะกับงานได้หลายๆแบบ และ Group ที่ใช้จะสร้างความยืดหยุ่น ช่วยให้ผู้ใช้สามารถเลือกปรับโครงสร้างตารางได้ตามใจต่อไปได้อีกดีกว่าการใช้ VBA จัดโครงสร้างตารางแล้วไม่ยืดหยุ่น เนื่องจากจัดได้แบบเดียวตามตัวรหัสที่เขียนไว้แล้วเท่านั้น

ต้องการ Insert Row เพื่อให้ข้อมูลเดิมที่ติดกัน กลายเป็นเว้นบรรทัด

สมมติว่าเซลล์ A1:A5 มีข้อมูลเป็น A, B, C, D, E ตามลำดับ ลองคิดหาทางทำให้ข้อมูลเดิมที่อยู่ใน row ติดกันกลายเป็นห่างกัน row เว้น row จะใช้วิธีใดดี ซึ่งมีข้อแม้ว่า วิธีที่ใช้นั้นต้องสามารถนำมาใช้กับตารางที่มีข้อมูลนับหมื่นๆ row ได้ทันทีด้วย

ปัญหานี้ไม่จำเป็นต้องใช้ VBA แม้แต่น้อย เพียงแค่ใช้คำสั่ง Data > Sort ก็เสร็จแล้ว ทำไมคำสั่งจัดเรียงจะช่วย Insert row ให้ได้ ลองทำตามนี้

ในเซลล์ด้านขวาของตาราง A1:A5 เดิม คือในเซลล์ B1:B10 ให้ใส่ตัวเลข 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 ตามลำดับ จากนั้นเลือกพื้นที่ A1:B10 แล้วสั่ง Data > Sort โดยจัดเรียงลำดับตามตัวเลข จะพบว่า ตารางเดิม A1:A5 ถูกแทรก row ระหว่างค่าเดิมให้ทันที เนื่องจากเซลล์ตัวเลขเมื่อจัดเรียงจากน้อยไปมากเป็น 1, 1, 2, 2, 3, 3, 4, 4, 5, 5 จะพาเซลล์ช่องว่างใน Column A ติดไปด้วย จึงดูเหมือน ตารางเดิมถูก Insert Row

อยากควบคุมการกรอกค่าลงไปตาราง ให้บันทึกได้เฉพาะเซลล์ที่ต้องการ

พอตารางข้อมูลใหญ่ขึ้น ซับซ้อนมากขึ้น คุณคงอยากหาทางทำให้ผู้ใช้แฟ้ม เกิดความสะดวกในการกรอกค่าลงไปตาราง ซึ่งบางคนอาจคิดไกลไปใช้ VBA สร้างจอสแสดงแบบฟอร์มขึ้นมาให้ผู้ใช้กรอกข้อมูล แล้วจึงส่งค่าที่กรอกในแบบฟอร์มไปยังเซลล์ที่ต้องการต่ออีกที

ปัญหานี้นอกจากจะใช้ Custom Views ช่วยปรับโครงสร้างตารางให้ง่ายขึ้นแล้ว แนะนำให้เลือกเซลล์ที่ยอมให้บันทึกข้อมูลทับได้ แล้วสั่ง **Format > Cells > Protection > ดัดเครื่องหมายกาช่อง Locked** ทิ้งไปแล้วตามด้วยคำสั่ง **Tools > Protection > Protect Sheet** จะพบว่าเมื่อกดปุ่ม Tab บนแป้นพิมพ์ Excel จะย้ายตำแหน่งเซลล์กระโดดไปตามเซลล์ที่เลือกตัดกาช่อง Lock ทิ้งให้ไปเรื่อยๆ

เมื่อต้องการบันทึกข้อมูล ให้กดปุ่ม Tab แล้วพิมพ์ข้อมูลบันทึกลงไป แล้วกดปุ่ม Tab แล้วพิมพ์ ทำเช่นนี้ซ้ำต่อไปจนบันทึกข้อมูลครบทุกเซลล์

อยากสั่งให้ Excel ทำงานซ้ำ

เพียงแค่กดปุ่ม F4 จะพบว่า Excel ทำงานทวนซ้ำคำสั่งล่าสุดให้เองทุกครั้งทีกดปุ่ม F4

ตัวอย่างข้างต้นนี้เป็นเพียงบางส่วนที่แสดงให้เห็นว่า เราสามารถใช้เพียงคำสั่งบนเมนู หรือใช้สูตรคำนวณ หรือใช้ทั้งคำสั่งบนเมนูมาใช้ร่วมกับสูตร เพื่อให้ Excel ทำงานได้เกินกว่าที่คิดกัน จึงขอแนะนำว่า ก่อนที่จะคิดใช้ VBA ควรเรียนรู้สักนิดว่า บนเมนู และสูตร Excel ยังมีอะไรที่ซ่อนไว้อีกบ้าง

คำสั่งบนเมนู

- Data > Consolidate เพื่อสร้างตารางยอดรวมจากหลายตาราง
- Data > Filter เพื่อแสดงรายการข้อมูลที่ต้องการ
- Data > SubTotal เพื่อสร้างตารางแบ่งยอดรวมเป็นยอดย่อย
- Data > Validation > List เพื่อสร้างปุ่มให้คลิกเลือกข้อมูล
- Data > What-If Analysis > Data Table เพื่อสรุปการคำนวณเมื่อตัวแปรมีการเปลี่ยนแปลง
- Data > Text to Columns เพื่อแยกค่าออกจากประโยค
- Insert > PivotTable เพื่อสรุปข้อมูลนำมาสร้างตารางเปรียบเทียบ
- Home > Find & Select เพื่อค้นหาหรือแก้ไขข้อมูล
- Data > Edit Links เพื่อจัดการกับสูตร link ข้ามแฟ้ม
- F2 F3 F4 F5 F9 ปุ่มลัด
- Format > Cells เพื่อกำหนดรูปแบบ
- Home > Conditional Formatting เพื่อกำหนดรูปแบบตามเงื่อนไข
- Home > Style เพื่อกำหนดรูปแบบหลักให้กับแฟ้ม
- Formulas > Define Name เพื่อดังชื่อให้กับตารางหรือให้กับสูตร

- Data > What-If Analysis > Goal Seek เพื่อคำนวณย้อนกลับไปหาตัวแปร หรือ Scenarios เพื่อเลือกใช้ตัวแปรคำนวณตามเหตุการณ์

ปุ่มบนแป้นพิมพ์

- Ctrl+* เพื่อเลือกพื้นที่ตารางที่ติดต่อกัน (Current Region)
- Ctrl+จุด เพื่อตรวจสอบห้วงมุมตารางที่เลือก
- Ctrl+Shift+ลูกศร เพื่อเลือกพื้นที่ตารางตามแนวที่ต้องการ
- Ctrl+PageUp เพื่อไปยังชีทก่อน
- Ctrl+PageDown เพื่อไปยังชีทถัดไป
- Ctrl+c = Copy
- Ctrl+x = Cut
- Ctrl+v = Paste
- Ctrl+z = Undo
- Shift+End,ลูกศร เพื่อเลือกพื้นที่ตารางตามแนวที่ต้องการ

สูตรสำเร็จรูป

If And Or Choose Vlookup Match Index CountIF SumIf SumArray Offset (จะนำมาอธิบายรายละเอียดในภายหลัง)

ต่อเมื่อคุณรู้จักคำสั่งบนเมนู วิธีใช้แป้นพิมพ์ลัด และสูตรพอควรแล้ว ควรทดลองใช้ร่วมกับ Macro Recorder จะช่วยประหยัดเวลาไม่ต้องเขียนรหัส VBA ได้มากที่สุดทีเดียว

เคล็ดลับการใช้ Macro Recorder

Macro Recorder เป็นเครื่องมือช่วยสร้างรหัส VBA ให้เอง ซึ่งรหัส VBA ที่เกิดขึ้นนั้น บางครั้งเป็นรหัสที่แม้แต่คนที่เก่ง VBA มากก็ยังไม่ถึง และยังเป็นเครื่องมือช่วยในการเรียนรู้ VBA แต่โดยทั่วไปมักกล่าวกันว่า เราสามารถนำ Macro Recorder มาใช้กับงานง่ายๆได้เท่านั้น หากเป็นงานยากๆยิ่งๆก็ยิ่งต้องหันมาเขียนรหัสอยู่ดี เนื้อหาในส่วนนี้จะแนะนำเคล็ดลับการใช้ Macro Recorder ให้เหนือกว่าที่คิดกัน

คลิก อันตราย

พยายามหลีกเลี่ยงการใช้ Macro Recorder บันทึกการใช้เมาส์คลิก ไม่ว่าจะคลิกเลือก เซลล์ คลิกเลือกชีท หรือแม้แต่การคลิกเลือกคำสั่งบนเมนู เพราะถ้าคลิกพลาด ตัว Macro Recorder จะบันทึกการทำงานพลาดของเราไปด้วย

ถ้าคลิกพลาดไปแล้ว Macro Recorder ก็จะสร้างรหัสซึ่งทำหน้าที่ตามที่พลาดตาม แล้วถ้าคลิกต่อไปเพื่อคลิกแก้ไขให้ถูก ชุดรหัสที่บันทึกจึงมีความยาวกว่าจำเป็น เพราะบันทึกตอนที่เรทำพลาด และบันทึกตอนที่แก้ไขต่อไปด้วย

ดังนั้นก่อนที่จะเริ่มบันทึก เราควรฝึกคลิกสิ่งที่ต้องการให้คล่องก่อน เมื่อบันทึกจริงจะได้ไม่พลาด

ยิ่งกว่านั้นถ้าไม่จำเป็นแล้ว อย่าใช้วิธีคลิก แต่ให้ใช้วิธีกดปุ่มบนแป้นพิมพ์แทน เพื่อลดโอกาสที่อาจจะคลิกพลาด เพราะปุ่มบนแป้นพิมพ์เห็นชัด และมีขั้นตอนเดียวตรงกับคำสั่งที่ต้องการ เช่น

- กดปุ่ม Ctrl+c, Ctrl+x, หรือ Ctrl+v เพื่อสั่ง copy, cut, หรือ paste
- กดปุ่ม F5 เพื่อเลือกไปที่ Range Name ที่ตั้งไว้
- กดปุ่ม Ctrl+PageUp เพื่อเลื่อนไปเลือกชีทก่อนหน้า
- กดปุ่ม Ctrl+PageDown เพื่อเลื่อนไปเลือกชีทถัดไป

ตัวอย่างเช่น หากต้องการ copy ข้อมูลจากตารางที่มี Range Name ชื่อ Source ไป paste ลงไปในตารางชื่อ Target ซึ่งมีพื้นที่เซลล์ขนาดเท่ากัน ให้กำหนดขั้นตอนการบันทึก Macro ดังนี้

1. กดปุ่ม F5 เลือกตารางชื่อ Source
2. กดปุ่ม Ctrl+c เพื่อสั่ง Copy
3. กดปุ่ม F5 เลือกตารางชื่อ Target
4. กดปุ่ม Ctrl+v หรือกด Enter เพื่อสั่ง Paste

จะเกิดชุดคำสั่งที่มีรหัสตรงตามแต่ละขั้น ดังนี้

```
Application.Goto Reference:="Source"
Selection.Copy
Application.Goto Reference:="Target"
ActiveSheet.Paste

เมื่อปรับปรุงรหัสเองให้สั้นลง
Range("Source").Copy Range("Target")
```

เตรียมทุกอย่างให้ตรงข้ามกับสิ่งที่อยากคลิก

ขอให้เตรียมทำทุกอย่างซึ่งตรงข้ามกับคำสั่ง VBA ที่ต้องการไว้ก่อน เพื่อจะได้คลิกกลับมาเป็นสิ่งที่ต้องการ เช่น ถ้าอยากบันทึก Macro การคลิกเลือกเซลล์ A1 ดังนั้นก่อนที่จะบันทึก Macro ต้องคลิกเลือกเซลล์อื่นที่ไม่ใช่เซลล์ A1 แล้วตอนที่บันทึก Macro จะได้เห็นได้ชัดว่า เราคลิกเข้าไปที่เซลล์ A1 จริง และจะเกิดรหัสขึ้นตามนั้น

```
Range("A1").Select
```

ถ้าอยากได้ Macro บันทึกการเลือก Sheet3 ดังนั้นก่อนที่จะเริ่มบันทึก Macro ให้เลือกชีทอื่นไว้ก่อน

```
Sheets("Sheet3").Select
```

ถ้าอยากได้ Macro บันทึกการเลือกแฟ้มชื่อที่ต้องการ ดังนั้นก่อนที่จะเริ่มบันทึก Macro ให้เปิดแฟ้มแล้วเลือกอยู่ในแฟ้มอื่นไว้ก่อน

```
Windows("Book1").Activate
```

ถ้าอยากได้ Macro บันทึกการปรับระบบการคำนวณให้เป็น Manual ดังนั้นก่อนที่จะบันทึก Macro ให้ปรับระบบให้เป็น Automatic หรือตัวเลือกอื่นไว้ก่อน

```
With Application
```

```
.Calculation = xlManual
```

```
.MaxChange = 0.001
```

```
End With
```

เมื่อปรับปรุงรหัสเองให้สั้นลง

```
Application.Calculation = xlManual
```

คลิกเพื่อสร้างรหัสที่จำกัดตายตัว หรือไม่คลิกเพื่อสร้างรหัสยืดหยุ่น

เมื่อใดที่คลิกลงไปชีทหรือเซลล์ เมื่อนั้น Macro Recorder จะสร้างรหัส VBA ที่ระบุชื่อชีทหรือตำแหน่งเซลล์ตายตัว เช่น หากต้องการควบคุมให้บันทึกเลข 123 ลงไปที่เซลล์ A5 ของ Sheet3 ให้เลือกเซลล์ที่อยู่ในชีทอื่นก่อน จากนั้นให้เริ่มบันทึก Macro โดยคลิกเลือก Sheet3 แล้วคลิกเซลล์ A5 พิมพ์เลข 123 ลงไปแล้วกด Enter ซึ่งจะเกิดรหัส VBA ซึ่งจะทำงาน ณ ตำแหน่งเซลล์เดิมของชีทชื่อเดิมด้วยเท่านั้น

```
Sheets("Sheet3").Select
```

```
Range("A5").Select
```

```
ActiveCell.FormulaR1C1 = "123"
```

ตัวอย่างนี้หากใช้วิธีที่ไม่คลิก โดยให้คลิกเลือกเซลล์ A5 ไว้ก่อน แล้วเริ่มบันทึก Macro เพียงขั้นเดียว โดยพิมพ์เลข 123 ลงไปแล้วกด Enter จะเกิดรหัสซึ่งสามารถนำไปใช้ได้กับทุกตำแหน่งเซลล์

```
ActiveCell.FormulaR1C1 = "123"
```

ฝึกใช้คำสั่งลดจากแป้นพิมพ์ เพื่อสร้างรหัสที่ยืดหยุ่น

ดังที่อธิบายไว้ในเบื้องต้นแล้วว่า วิธีคลิกเมนู คลิกเซลล์ หรือคลิกเพื่อเลือกสิ่งที่ต้องการบนหน้าจอ จะสร้างรหัสที่ตายตัว ไม่สามารถนำรหัสไปใช้ต่างที่ต่างเวลา แตกต่างจากสถานการณ์เดิม แทนที่จะคลิก ขอให้ใช้แป้นพิมพ์ในระหว่างการบันทึก Macro ซึ่งทั้งนี้วิธีใช้แป้นพิมพ์ที่ดูแล้วว่า ทำงานเหมือนกัน ไม่จำเป็นว่าจะเกิดรหัสจากการบันทึกที่ยืดหยุ่นเสมอไป

สมมติว่า ในตารางมีข้อมูลบันทึกไว้ในเซลล์ A1:D2 แล้วเราต้องการสร้างรหัส VBA เพื่อเลือกพื้นที่ที่เก็บข้อมูล จะสามารถใช้แป้นพิมพ์เลือกพื้นที่เดียวกันนี้ได้หลายแบบ และรหัสแต่ละแบบจะมีความยืดหยุ่นมากน้อยต่างกัน

	A	B	C	D	E
1	11	22	33	44	
2	55	66	77	88	
3					

ก่อนที่จะเริ่มบันทึก Macro ขอให้คลิกเลือกเซลล์ A1 ไว้ก่อนจากนั้นจึงเริ่มบันทึก Macro ตามการใช้แป้นพิมพ์ได้หลายวิธี

วิธีที่ 1 กดปุ่ม Ctrl+* เพื่อเลือก Current Region รหัส VBA ที่เกิดขึ้นคือ

```
Selection.CurrentRegion.Select
```

วิธีที่ 2 ต้องบันทึก 2 ขั้นตอน

1. กดปุ่ม Ctrl+Shift+ลูกศรขวา พร้อมกัน จากนั้น
2. กดปุ่ม Ctrl+Shift+ลูกศรขึ้น พร้อมกัน

```
Range(Selection, Selection.End(xlToRight)).Select
```

```
Range(Selection, Selection.End(xlUp)).Select
```

วิธีที่ 3 ต้องใช้แป้นพิมพ์เลือกทีละเซลล์จากขวามาซ้าย ตามด้วยจากบนมาล่าง

1. กดปุ่ม Shift พร้อมกับกดปุ่มลูกศรขวา 3 ครั้ง
2. กดปุ่ม Shift พร้อมกับกดปุ่มลูกศรลง 1 ครั้ง

```
Range("A1:D2").Select
```

วิธีที่ 4 กดปุ่ม **Ctrl+Shift+End** เพื่อเลือกจนถึงเซลล์ขวาล่างสุดของตาราง

```
Range(Selection, _  
ActiveCell.SpecialCells(xlLastCell)).Select
```

วิธีที่ 5 วิธีนี้ใช้หลักเลือกย้อนจากล่างขึ้นบน

1. กดปุ่ม F5 ไปเลือกเซลล์ D65536 บรรทัดล่างสุดของตาราง
2. กดปุ่ม End แล้วตามด้วยปุ่มลูกศรชี้ขึ้น เพื่อเลือกเซลล์ D2
3. กดปุ่ม Ctrl+Shift พร้อมกับปุ่ม Home เพื่อเลือกย้อนกลับมาเซลล์ A1

```
Application.Goto Reference:="R65536C4"  
Selection.End(xlUp).Select  
Range(Selection, Cells(1)).Select
```

ทั้ง 5 วิธีสามารถใช้เลือกพื้นที่ตารางจากเซลล์ A1:D2 คือ เซลล์ที่มีเลข 11 จนถึงเซลล์ที่มีเลข 88 ได้เหมือนกัน แต่วิธีที่ยืดหยุ่นคือวิธีที่ 5 วิธีเดียวเท่านั้น เพราะสามารถนำมาใช้เลือกพื้นที่ จากเซลล์ที่มีเลข 11 จนถึงเซลล์ที่มีเลข 88 ได้เหมือนเดิม แม้ว่าตารางจะมีลักษณะเปลี่ยนไป ไม่ได้มีข้อมูลในเซลล์ติดต่อกันเช่นเดิม

	A	B	C	D	E
1	11		33	44	
2					
3		66	77	88	
4					
5					XXXX

ตัวอย่างนี้แสดงให้เห็นว่า เป็นหน้าที่ของผู้ใช้ Excel ที่จะต้องเลือกใช้วิธีใช้แป้นพิมพ์อย่างชาญฉลาดด้วย จึงจะทำให้ Macro Recorder สร้างรหัส VBA ที่ยืดหยุ่น สามารถนำมาใช้กับงานที่มีโครงสร้างข้อมูลแตกต่างไปจากเดิม โดยที่เราไม่ต้องเสียเวลากลับมาเขียนปรับปรุงแก้ไขรหัสในภายหลัง

ใช้ Range Name แทนตำแหน่งอ้างอิงโดยตรง

รหัสที่ถูกบันทึกหรือเขียนไว้ใน VBE จะคงที่ตามเดิมไปตลอด ไม่มีการเปลี่ยนแปลงแก้ไขส่วนหนึ่งส่วนใดอีกต่อไป จนกว่าเราจะเข้าไปแก้ไขเอง ซึ่งเป็นจุดอ่อนที่สำคัญของตัว VBE ทำให้เกิดปัญหาแทบทุกครั้งที่เราสร้างข้อมูล ชื่อชีท หรือตำแหน่งเซลล์ในตารางเปลี่ยนที่ไปจากเดิม ส่งผลให้ VBA ทำงานผิดพลาด จนกว่าเราจะเข้าไปปรับปรุงรหัสที่อ้างถึงตำแหน่งต่างๆ ในตารางตามจริงอีกครั้ง

ตามตัวอย่างข้างต้น หากต้องการใช้ VBA เลือกพื้นที่ตั้งแต่เซลล์ที่มีเลข 11 จนถึงเซลล์ที่มีเลข 88 ให้ใช้เมนู **Formulas > Define Name** ตั้งชื่อ Range **A1:D2** สมมติว่าใช้ชื่อ Range Name ว่า **MyData**

จากนั้นให้ใช้ Macro Recorder บันทึกการกดปุ่ม **F5** แล้วเลือกชื่อ **Range Name** ชื่อ **MyData** ที่แสดงขึ้น จะทำให้เกิดรหัส VBA ดังนี้

```
Application.Goto Reference:="MyData"
```

ซึ่งจะพบว่ารหัสเพียงบรรทัดเดียวนี้สามารถใช้งานได้ตลอดไป โดยไม่ต้องเสียแรงเสียเวลากลับมาแก้ไขอะไรอีก ไม่ว่าจะเป็นการย้ายตำแหน่งข้อมูลจากเดิม A1:D2 ไปอยู่ที่เซลล์อื่น ชีทอื่น หรือแฟ้มอื่น หรือแม้ว่าตารางที่เก็บข้อมูลเดิมนั้น ไม่ได้มีเซลล์เก็บข้อมูลต่อเนื่องกันเช่นเดิมแล้วก็ตาม

บันทึกสั้นๆ แล้วนำมาต่อกันให้ยาวขึ้น

ในการบันทึก Macro แต่ละครั้ง ขอให้บันทึกขั้นตอนทำงานสั้นๆ พยายามบันทึกขั้นตอนให้สั้นที่สุด เพื่อทำให้เกิดรหัสแต่ละงานถูกบันทึกเป็น Sub Procedure แยกออกจากกัน ช่วยทำให้เราสามารถเรียนรู้และแก้ไขรหัสได้ง่าย ต่อเมื่อผ่านการทดลองทดสอบแล้วว่า รหัสทั้งหมดสามารถทำงานได้ตามต้องการ จึงนำชื่อ Sub Procedure ที่ได้นั้นมาเขียนตามลำดับที่ใช้งาน รวมไว้ใน Sub Procedure ชุดใหม่ เพื่อช่วยทำให้เราสามารถสั่งงานเพียงครั้งเดียว แล้วรหัสทั้งหมดจะทำงานต่อเนื่องกันให้เอง

ตัวอย่างเช่น เดิมมี Sub Procedure ที่เกิดจากการบันทึก 3 ครั้ง ตั้งชื่อ Macro Name ไว้ว่า MyWork1, MyWork2, และ MyWork3

ให้เขียน Sub Procedure ใหม่เพื่อรวมคำสั่งเดิมทั้ง 3 ชุดมาไว้ที่เดียวกัน ตามแบบดังนี้

```
Sub RunAllMyWorks()
```

```
    MyWork1
```

```
    MyWork2
```

```
    MyWork3
```

```
End Sub
```

วิธีใช้ Macro Recorder ช่วยในการเรียนรู้ VBA

คนที่เพิ่งเริ่มฝึกใช้ VBA หรือแม้แต่คนที่ใช้ VBA มานาน มักติดปัญหาตรงที่ไม่รู้ว่า รหัสคำสั่ง VBA ที่ตนเองนำมาใช้นั้นเป็นอย่างไร บางคนสามารถเปิดตำราค้นหารหัสได้แล้ว แต่ไม่รู้ว่าจะดัดแปลงรหัสอย่างไรได้บ้าง ครั้นติดปัญหา ไม่สามารถแก้ไขทำต่อด้วยตนเอง ก็จะยกปัญหาขึ้นถามบนเว็บ พอได้คำตอบไปใช้งานสักพัก ต่อมาไม่นานก็กลับมาถามหารหัสต่อจากคำถามเดิมซ้ำแล้วซ้ำอีก

ขอย้ำอีกครั้งว่าถ้าอยากคิดใช้ VBA ต้องขอให้ตนเองเป็นที่พึ่งแห่งตน เพราะ VBA มีทั้งคุณ มีทั้งโทษ เป็นดาบสองคม หากคิดใช้ VBA แล้วต้องขอให้ตนเองหาทางใช้ให้ถูกวิธี ทุกครั้งที่ติดปัญหา ต้องพยายามหาทางแก้ไขด้วยตนเองให้ได้ ต่อเมื่อหมดหนทางแล้วจึงสอบถามจากผู้มีความรู้ และทำความเข้าใจกับคำตอบที่ได้รับมา อย่าท้อ อย่าลอก อย่าเอาไปใช้โดยปราศจากความเข้าใจ

ในบทนี้จะแนะนำวิธีใช้ Macro Recorder ช่วยในการเรียนรู้ VBA จะได้ทำตนให้เป็นที่พึ่งของตนเองเสียก่อน

ตัวอย่างที่ 1 : ต้องการเปลี่ยนชื่อชีท ให้มีชื่อใหม่ตามค่าที่อยู่ในเซลล์ A1

สมมติ เดิมชีทมีชื่อว่า Sheet1 เราต้องการเปลี่ยนชื่อชีทตามค่าที่เก็บไว้ในเซลล์ A1 เช่น ถ้า A1 มีคำว่า MyJan ก็ให้รหัส VBA ทำหน้าที่เปลี่ยนชื่อชีทเป็น MyJan และถ้า A1 มีค่าใหม่ ก็ให้ชื่อชีทถูกเปลี่ยนชื่อใหม่ตามค่าในเซลล์ A1 ได้ตามต้องการ

ขั้นที่ 1 : ใช้ Recorder บันทึกการคลิกเลือกเซลล์ A1 แล้วพิมพ์คำว่า MyJan ลงไป จะได้รหัส VBA :

```
Range("A1").Select
ActiveCell.FormulaR1C1 = "MyJan"
```

ขั้นที่ 2 : ใช้ Recorder บันทึกการสั่ง **Home > Format > Rename Sheet** แล้วพิมพ์คำว่า MyJan ทับคำว่า Sheet1 จะได้รหัส VBA :

```
Sheets("Sheet1").Name = "MyJan"
```

สังเกตว่าผลที่ได้จากการบันทึกขั้นตอนนี้ แสดงว่าการกำหนดค่าให้กับสิ่งใด ให้ใช้เครื่องหมายเท่ากับ โดยสิ่งที่อยู่ด้านซ้ายของเครื่องหมายเท่ากับ ถือเป็นสิ่งที่ถูกกำหนดค่า ให้มีค่าตาม ค่าที่อยู่ด้านขวาของเครื่องหมายเท่ากับ

ขั้นที่ 3 : ทดลองเขียน VBA โดยผสมรหัสทั้งสองขั้นเข้าด้วยกันเป็นดังนี้

```
Sub RenameSheet()  
    Sheets("Sheet1").Name = Range("A1")  
End Sub
```

ซึ่งจะพบว่ารหัสใน Sub Procedure ชื่อ RenameSheet นี้ทำงานได้เพียงครั้งเดียว เนื่องจากตัวรหัส Sheets("Sheet1").Name กำหนดไว้ว่า ให้เปลี่ยนชื่อที่มีชื่อว่า Sheet1 ดังนั้นหากในแฟ้มไม่มีชื่อ Sheet1 เพราะถูกเปลี่ยนชื่อชื่อเป็น MyJan ไปแล้ว รหัสจึงไม่สามารถทำงานตามต้องการ

ขอให้สังเกตย้อนกลับไปดูรหัสที่ได้จากขั้นแรก จะเห็นว่ามียุทธศาสตร์ว่า ActiveCell ซึ่งมีหมายถึงเซลล์ตำแหน่งใดก็ได้ ขอเพียงแต่กำลังถูกเลือกใช้งานอยู่ จึงมีคำว่า Active นำหน้า สงสัยไหมว่าในเรื่องของชื่อนั้นมี ActiveSheet ด้วยไหม คราวนี้ให้ลองใช้คำว่า ActiveSheet ค้นหาใน Help คุณจะพบคำอธิบาย ดังนี้

ActiveSheet Property

Returns an object that represents the active sheet (the sheet on top) in the active workbook or in the specified window or workbook. Returns Nothing if no sheet is active. Read-only.

Remarks

If you don't specify an object qualifier, this property returns the active sheet in the active workbook.

If a workbook appears in more than one window, the ActiveSheet property may be different in different windows.

Example

This example displays the name of the active sheet.

```
MsgBox "The name of the active sheet is " & ActiveSheet.Name
```

ถ้าอยากอ่านคำอธิบายที่ละเอียดก็ตามใจ แต่ส่วนสำคัญที่สุดที่ควรดูเป็นจุดแรกก็คือ ตัวอย่างที่แสดงไว้ จะเห็นว่ามีคำสั่ง `ActiveSheet.Name` ซึ่งแปลว่า ชื่อชีทของชีทที่กำลังเลือกอยู่ และน่าจะนำมาใช้แทนคำว่า `Sheets("Sheet1").Name` ได้

ขั้นที่ 4 : กลับไปแก้รหัสที่เขียนเองใหม่ แล้วจะพบว่ารหัสชุดนี้ทำงานได้ตลอด

```
Sub RenameSheet()  
    ActiveSheet.Name = Range("A1")  
End Sub
```

หมายเหตุ ถ้าค้นหาคำว่า Active จาก Help จะพบว่านอกจาก `ActiveCell` และ `ActiveSheet` แล้วยังมี `ActiveWorkbook`, `ActiveWindow`, `ActiveChart`, `ActivePane`, และ `ActivePrinter` ให้เลือกนำมาใช้ได้

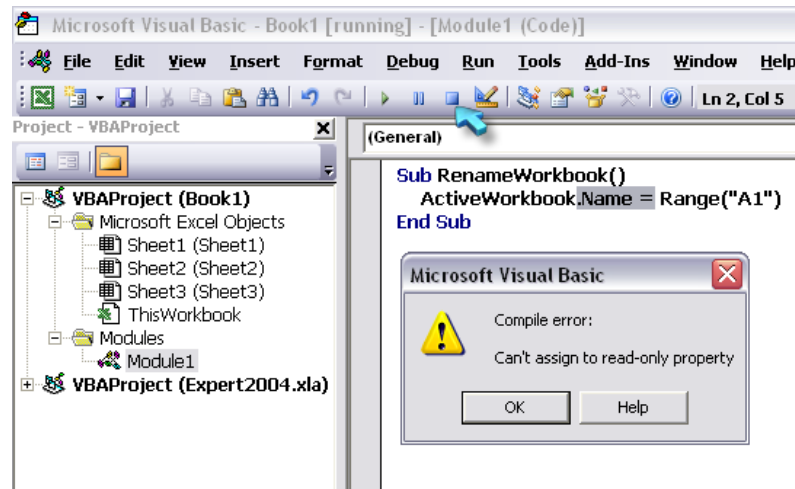
ตัวอย่างที่ 2 : ต้องการจัดเก็บแฟ้ม โดยใช้ชื่อแฟ้มตามค่าที่อยู่ใน เซลล์ A1

ตัวอย่างนี้ต้องการให้รหัสทำงานคล้ายกับตัวอย่างแรก เพียงแต่คราวนี้ให้เปลี่ยนชื่อแฟ้มแทนที่จะเป็นชื่อชีท

ขั้นที่ 1 : ขอให้ลองเขียนรหัสเองขึ้นมา โดยเลียนแบบตัวอย่างแรก

```
Sub RenameWorkbook()  
    ActiveWorkbook.Name = Range("A1")  
End Sub
```

เมื่อสั่ง Run จะพบว่ารหัสชุดนี้ไม่ทำงาน โดยมีคำเตือนว่า **Compile error** ในขั้นนี้ไม่ต้องสนใจว่า `Compile error` คืออะไร เพียงขอให้สังเกตว่า รหัสไม่ทำงาน แสดงว่าต้องมีอะไรที่ผิดพลาด และถ้าสังเกตให้ดีจะพบว่าเมื่อคลิก OK ออกจากคำเตือน `Compile error` แล้ว VBE จะเลือกคำว่า `.Name` ไว้ เป็นอาการที่ชี้ให้เราเห็นว่า ส่วนที่ผิดพลาดน่าจะเป็นส่วนของคำว่า `.Name` นี้เอง



ทุกครั้งที่เราสั่งไม่ทำงาน ให้คลิกปุ่ม Reset ซึ่งเป็นปุ่มสี่เหลี่ยมบนเมนูก่อน มิฉะนั้นจะทำงานอื่นต่อไปไม่ได้เลย

ขั้นที่ 2 : ใช้ Recorder บันทึกการสั่ง **File > Save as** เป็นชื่อแฟ้มใหม่ สมมติให้ใช้ชื่อว่า MyMonth.xls จะได้รหัส VBA :

```
ActiveWorkbook.SaveAs Filename:= _
"C:\MyMonth.xls", FileFormat:= _
xlNormal, Password:="", WriteResPassword:="", _
ReadOnlyRecommended:=False _
, CreateBackup:=False
```

ขั้นที่ 3 : ทดลองตัดรหัสในขั้นที่ 2 ให้สั้นลงเหลือเท่าที่ตรงกับการจัดเก็บแฟ้มตามชื่อที่ต้องการ แล้วทดสอบว่ารหัสใหม่นี้ทำงานได้หรือไม่ ซึ่งจะพบว่ารหัสทำงาน เพียงแต่จะถูกถามว่า ต้องการให้ต้องการจัดเก็บในชื่อนี้หรือไม่ เพราะมีแฟ้มชื่ออยู่แล้ว

```
ActiveWorkbook.SaveAs Filename:="MyMonth.xls"
```

ขั้นที่ 4 : ย้อนกลับไปแก้รหัสในขั้นที่ 1 แล้วจะพบว่ารหัสทำงานได้ตามต้องการ

```
Sub RenameWorkbook()
    ActiveWorkbook.SaveAs Filename: = Range("A1")
End Sub
```

ตัวอย่างที่ 3 : ต้องการจัดปิดแฟ้ม โดยไม่ต้องจัดเก็บซ้ำอีก

ขั้นที่ 1 : ตัวอย่างนี้ต้องเปิดแฟ้มใหม่ขึ้นอีกแฟ้มหนึ่ง เพื่อใช้เก็บรหัสที่ได้จากการบันทึกการปิดแฟ้ม โดยมีขั้นตอนใช้ Recorder บันทึก ดังนี้

1. คลิกเมนู View > Switch Windows > เลือกชื่อแฟ้มที่ต้องการปิด สมมติว่าชื่อ MyMonth.xls
2. คลิกเมนู File > Close

จะเกิดรหัส VBA :

```
Windows("MyMonth.xls").Activate
ActiveWorkbook.Close
```

ขั้นที่ 2 : ให้ copy รหัสเฉพาะ ActiveWorkbook.Close มาสร้าง Sub Procedure ลงใน MyMonth.xls

```
Sub CloseMyFile()
    ActiveWorkbook.Close
End Sub
```

เมื่อทดสอบรหัสนี้หลายๆครั้ง จะพบว่าถ้าในแฟ้มมีข้อมูลใหม่ จะมีคำเตือนเปิดขึ้นบนจอว่า คุณต้องการจัดเก็บข้อมูลที่เปลี่ยนแปลงหรือไม่

ขั้นที่ 3 : ถ้าอยากใช้คำสั่งปิดแฟ้ม โดยไม่ต้องถามเรื่องการจัดเก็บข้อมูลใหม่ คราวนี้ต้องศึกษาจาก Help โดยให้คลิกที่รหัสคำว่า Close จากนั้นกดปุ่ม F1 จะมีตัวอย่างแสดงไว้ในคำอธิบายของ Help ดังนี้

```
Workbooks("BOOK1.XLS").Close SaveChanges:=False
```

ขั้นที่ 4 : กลับไปแก้ไขรหัสเดิม โดยนำคำว่า SaveChanges:=False ไปเพิ่มต่อท้ายรหัสเดิม จะพบว่าคราวนี้ไม่มีคำเตือนเปิดขึ้นบนจอว่า คุณต้องการจัดเก็บข้อมูลที่เปลี่ยนแปลงหรือไม่

```
Sub CloseMyFile()
    ActiveWorkbook.Close SaveChanges:=False
End Sub
```

หมายเหตุ : หากศึกษาคำอธิบายจาก Help จะพบว่า คำสั่ง Close นั้น สามารถเขียนอีกรูปแบบหนึ่ง คือ *expression.Close(SaveChanges, Filename, RouteWorkbook)* โดยใช้วงเล็บกำหนดตัวเลือกต่างๆกันไป ทำให้รหัสสั้นลงเป็น

```
Sub CloseMyFile()
    ActiveWorkbook.Close(False)
End Sub
```

วิธีสร้างชุดคำสั่งควบคุมระบบเมื่อเปิดปิดแฟ้ม

ปัญหาหนึ่งซึ่งเกิดกับทุกคนที่ต้องแบ่งกันใช้เครื่องคอมพิวเตอร์เครื่องเดียวกันทำงาน มักพบว่า เมื่อเปิดโปรแกรม Excel ขึ้นมาแต่ละครั้ง จะพบหน้าต่างของเมนู หรือสภาพพื้นที่ของแฟ้มแรกกว้างๆ ที่เห็นบนหน้าจอ ขอบเปลี่ยนตำแหน่งหน้าต่างแตกต่างไปจากเดิมที่เคยใช้ เพราะเพื่อนคนที่ใช้ Excel คราวก่อน อาจปรับตำแหน่งเมนู และระบบต่างๆ ของ Excel ให้ถูกใจตน ... แต่ไม่ถูกใจคนอื่นเขา

ดังนั้นแทนที่จะต้องเสียเวลาปรับระบบของ Excel ให้ถูกใจ เรามาใช้ Macro Recorder บันทึกรหัสเพื่อทำหน้าที่ปรับระบบให้ทันทีเมื่อเปิดแฟ้มขึ้นมาดีกว่า พอเปิดแฟ้มขึ้นมา ก็ให้ระบบปรับตัวเองทันที และเมื่อปิดแฟ้มก็ให้ปรับระบบกลับสู่สภาพแรกสุด เหมือนสมัยที่เพิ่งเปิดใช้ Excel เป็นครั้งแรก

วิธีใช้ Macro Recorder ต่อไปนี้ มีข้อสำคัญอยู่ที่ชื่อของ Macro ว่าต้องตั้งชื่อว่า **Auto_Open** และ **Auto_Close** เท่านั้น ห้ามตั้งชื่อเป็นอย่างอื่น โดย Auto_Open จะถูกสั่งให้ทำงานเองทันทีเมื่อแฟ้มที่เก็บรหัสนี้ถูกเปิดขึ้น และ Auto_Close จะถูกสั่งให้ทำงานทันทีเมื่อแฟ้มที่เก็บรหัสนี้ถูกสั่งปิด

ก่อนที่จะเริ่มบันทึก Macro อย่าลืมปรับระบบ Excel ให้เป็นตรงกันข้ามกับที่ต้องการไว้ก่อน เพราะระหว่างการบันทึก Macro นั้น เราจะได้เห็นว่าคำสั่งที่ต้องการนั้นถูกเลือกจริง และ Macro Recorder จะสร้างรหัสตาม

ขั้นตอนการบันทึก Macro ชื่อ Auto_Open

สมมติ เราต้องการปรับหน้าจอของแฟ้มที่เปิดขึ้นมาให้ดูเหมือนกระดาษทำการ หากทางทำให้ไม่มีลักษณะใดที่จะบ่งบอกว่าเป็นตารางพื้นที่ Worksheet ทั้งนี้อาจใช้กับการเสนอผลงาน Presentation หรือใช้กับงานทั่วไปที่ต้องการปิดบังไม่ให้ผู้ใช้ Excel สงสัยหรือรู้ว่า หน้าจอที่เปิดขึ้นนั้นเป็นตาราง Excel

(หลักการใช้ Excel ที่ดี เมื่อสร้างงานเสร็จแล้ว ควรปรับสภาพหน้าจอให้ดูเหมือนกระดาษ เพื่อให้ผู้ใช้งานรู้สึกเหมือนว่า ตนกำลังทำงานอยู่บนหน้ากระดาษที่ตนเองคุ้นเคย ไม่ใช่ตาราง Excel เพื่อให้ผู้ใช้เกิดความสบายใจมากขึ้น)

เริ่มต้นบันทึก Macro โดยกำหนดชื่อ Macro Name : **Auto_Open** และเลือกใช้ **Ctrl+q** หรือใช้ปุ่มอื่นก็ได้เป็น Shortcut Key แล้วบันทึกขั้นตอนคำสั่งต่อไปนี้

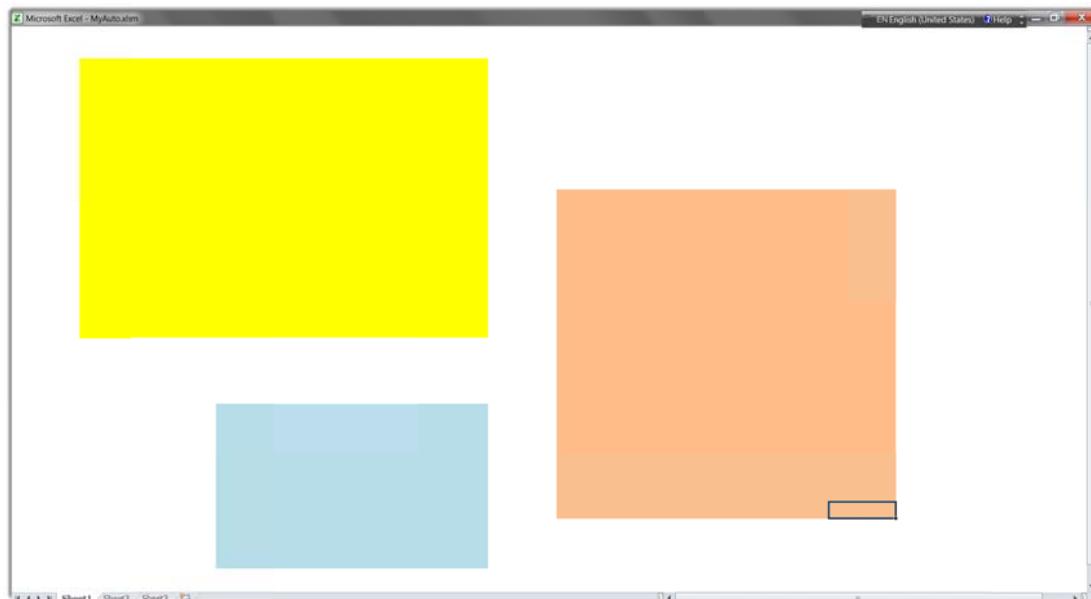
1. สั่ง Formulas > Calculation Options > กาช่อง Manual
2. สั่ง View > ตัดกาช่อง Gridlines เพื่อตัดเส้นตาราง
3. สั่ง View > ตัดกาช่อง Headings เพื่อตัดหัวตารางที่บอกว่าเป็น row และ column ไດ
4. สั่ง View > ตัดกาช่อง Formula Bar เพื่อช่องใส่สูตร
5. สั่ง View > Full Screen เพื่อตัดเมนูทั้ง ขยายพื้นที่แสดงตาราง
6. กดปุ่ม Esc เพื่อย้อนกลับมาสู่จอตามขนาดเดิมจะได้กดปุ่ม Stop ได้

ในชุดคำสั่งที่เกิดขึ้น ให้ตัดคำสั่งสุดท้ายทิ้งโดยทำให้เป็น Comment

```
Sub Auto_Open()
'
' Auto_Open Macro
'
' Keyboard Shortcut: Ctrl+q
'

Application.Calculation = xlManual
Application.DisplayFormulaBar = False
ActiveWindow.DisplayHeadings = False
ActiveWindow.DisplayGridlines = False
Application.DisplayFullScreen = True
` Application.DisplayFullScreen = False กำหนดคำสั่งนี้ให้เป็น comment
End Sub
```

เมื่อสั่งให้ Auto_Open ทำงาน จะปรับระบบการคำนวณเป็น Manual และหน้าจอให้ดูเหมือนกระดาษ



ขั้นตอนการบันทึก Macro ชื่อ Auto_Close

ขั้นตอนที่ใช้บันทึก Macro ในชื่อ Auto_Close นั้น จำง่าย ๆ ว่า ทำตามขั้นตอนของ Auto_Open เพียงแต่ให้ทำตรงกันข้ามให้หมด (หรือจะเพิ่มคำสั่งอื่นก็ได้ที่เห็นว่า ควรปรับระบบของ Excel ให้เหมาะกับงานทั่วไป)

ก่อนจะเริ่มบันทึก Macro ให้กดปุ่ม Esc เพื่อแสดงเมนูให้เห็น เมื่อบันทึกให้กำหนดชื่อ Macro Name : **Auto_Close** และเลือกใช้ **Ctrl+w** หรือใช้ปุ่มอื่นก็ได้เป็น Shortcut Key แล้วบันทึกขั้นตอนคำสั่งต่อไปนี้

1. สั่ง Formulas > Calculation Options > กาช่อง Automatic
2. สั่ง View > กาช่อง Gridlines เพื่อตัดเส้นตาราง
3. สั่ง View > กาช่อง Headings เพื่อตัดหัวตารางที่บอกว่าเป็น row และ column ไต
4. สั่ง View > กาช่อง Formula Bar เพื่อช่องใส่สูตร

ให้แก่ชุดคำสั่งที่เกิดขึ้น โดยลอกคำสั่ง Application.DisplayFullScreen = False มาใส่เพิ่ม

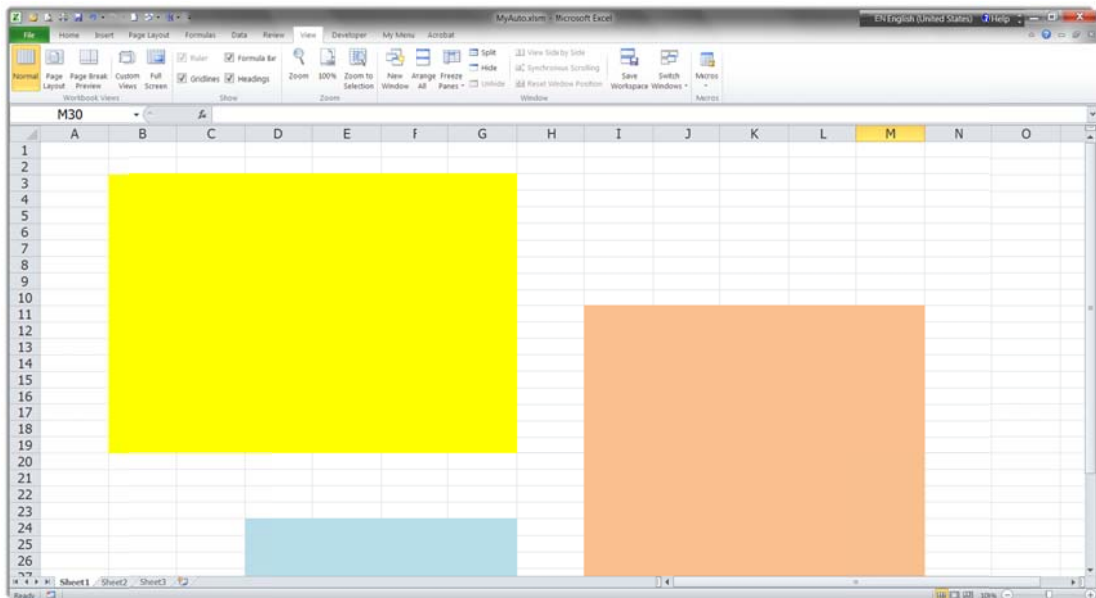
```

Sub Auto_Close()
'
' Auto_Close Macro
'
' Keyboard Shortcut: Ctrl+w
'

Application.DisplayFullScreen = False 'ลอกมาใส่เพิ่ม
Application.Calculation = xlAutomatic
Application.DisplayFormulaBar = True
ActiveWindow.DisplayHeadings = True
ActiveWindow.DisplayGridlines = True
End Sub

```

เมื่อสั่งให้ Auto_Close ทำงาน จะปรับระบบการคำนวณเป็น Automatic และหน้าจอกลับสู่สภาพเดิม



ทดลองแก้ไขรหัสให้ทำงานกลับไปกลับมาทุกครั้งที่เราสั่งให้ทำงาน

ขอให้สังเกตรหัสที่ได้จากการบันทึก Macro ข้างต้น จะเห็นว่าใช้คำสั่งให้เท่ากับ True หรือ False เพื่อสั่งให้ทำหรือไม่ทำตาม ซึ่งหมายถึงสั่งให้ทำตรงกันข้ามกับสถานะเดิม ดังนั้นเราจึงสามารถใช้รหัสคำสั่งต่อไปนี้ เพื่อปรับการแสดง Full Screen ให้กลับไปซ่อนเมนูแล้วกลับมาตามเดิมได้ โดยใช้คำสั่งบรรทัดเดียวและใช้รหัส NOT ช่วยให้ทำงานกลับกัน ดังนี้

```
Sub ReverseFullScreen()  
Application.DisplayFullScreen = Not Application.DisplayFullScreen  
End Sub
```

ทุกครั้งที่รหัสคำสั่งชุดนี้ทำงาน จะปรับหน้าจอให้ตรงกันข้ามกับสถานะเดิม เช่น ถ้าเดิมไม่ได้เป็น Full Screen พอใช้คำสั่งนี้จะกลายเป็น Full Screen แต่ถ้าสั่งซ้ำอีกครั้งจะกลับเป็นสภาพเดิมที่ไม่ได้เป็น Full Screen

รหัส VBA ที่ยืดหยุ่นตามการเปลี่ยนแปลงใน Excel

เมื่อเปรียบเทียบโปรแกรม Excel กับ VBE ในเรื่องความยืดหยุ่นแล้วจะพบว่า รหัสทั่วไปที่เขียนลงไปใน VBE จะไม่ปรับตัวเองให้สอดคล้องกับโครงสร้างตารางในชีทที่เปลี่ยนไป จึงถือเป็นจุดอ่อนสำคัญ และส่งผลเสียทำให้เราต้องเสียเวลาย้อนกลับมาตามแก้ไขรหัสซ้ำแล้วซ้ำอีกอยู่เสมอ

ลองพิจารณารหัสที่ได้จากการใช้ Macro Recorder บันทึกการไล่คลิกหาแฟ้ม ต่อด้วยคลิกหาชีท แล้วคลิกหาเซลล์ที่ต้องการพิมพ์เลข 123 ลงไป

```
Windows("Book1.xls").Activate
Sheets("Sheet1").Select
Range("A1").Select
ActiveCell.FormulaR1C1 = "123"
```

จะพบว่า กว่าจะกำหนดตำแหน่งเซลล์ A1 ที่ต้องการ จะต้องระบุชื่อแฟ้มและชื่อชีทลงไปในตัวรหัส แล้วต่อมาถ้าเซลล์เก็บข้อมูลถูกย้ายไปเซลล์อื่นแทน A1 จะส่งผลให้รหัสชุดนี้ไม่ทำงานตามเดิม

นอกจากรหัสที่ได้จาก Recorder แล้ว รหัสที่เขียนขึ้นเอง เพื่อทำหน้าที่เช่นเดียวกับชุดรหัสข้างต้น แม้มีรหัสสั้นลงเหลือเพียงบรรทัดเดียว แต่ยังจำเป็นต้องระบุชื่อแฟ้ม ชื่อชีท และตำแหน่งเซลล์ ซึ่งเป็นเหตุให้ไม่ยืดหยุ่นเช่นกัน

```
Workbooks("Book1.xls").
Worksheets("Sheet1").Range("A1") = 123
```

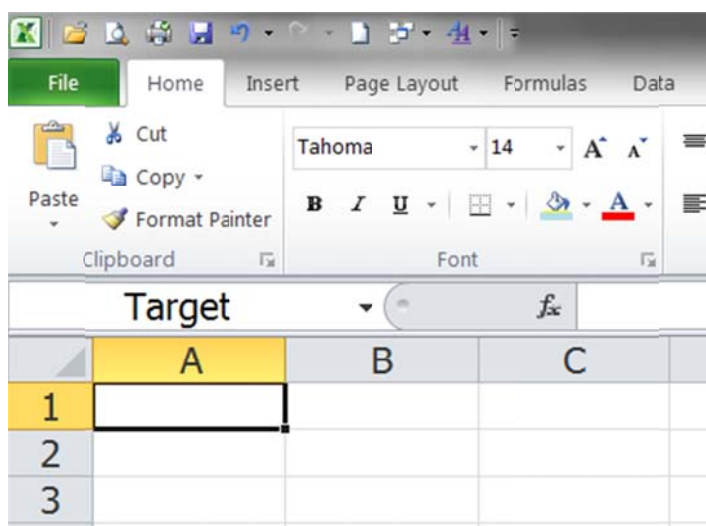
ขอให้สังเกตว่า รหัสที่ได้จาก Recorder และรหัสที่เขียนเองนี้ แม้จะทำให้ได้ค่า 123 ใส่ลงไปในเซลล์ A1 ได้เช่นเดียวกันก็ตาม แต่รหัสที่เขียนเองจะช่วยส่งเลข 123 ลงไปในเซลล์ A1 ให้โดยไม่จำเป็นต้องย้ายไปเลือกเซลล์ A1 ส่งผลให้ Excel ทำงานได้เร็วขึ้นด้วย เนื่องจากไม่ต้องเสียเวลาเลือกเซลล์ซึ่งเกิดจากคำสั่ง Select

วิธีใช้ Range Name เพื่อทำให้รหัส VBA เกิดความยืดหยุ่น

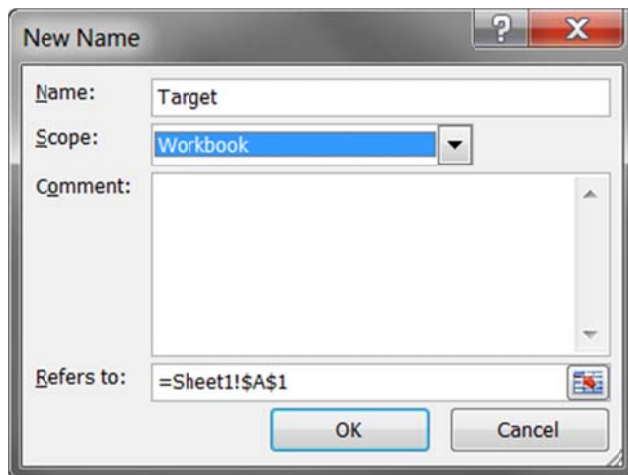
Range Name เป็นชื่อที่เราตั้งขึ้นให้กับเซลล์ โดยจะเป็นเซลล์เดียว หรือหลายเซลล์ก็ได้ ซึ่งชื่อ Range Name ที่ตั้งขึ้นนั้นนอกจากจะช่วยให้สูตรมีความหมายชัดเจนในตัวเองมากกว่าการใช้สูตรอ้างอิงจากเซลล์โดยตรง (เช่น เดิมใช้สูตรหากำไร =A1-B1 แต่ต่อมาใช้ Range Name แทนจะเห็นสูตร =Income-Cost) เรายังใช้ Range Name ช่วยทำให้รหัส VBA ที่มีอยู่นั้น ไม่จำเป็นต้องถูกแก้ไขเปลี่ยนแปลงอีกต่อไป

วิธีตั้งชื่อ Range Name ให้กับเซลล์ สามารถใช้วิธีง่ายๆโดยคลิกเลือกเซลล์หรือพื้นที่ตารางที่ต้องการก่อน จากนั้นให้พิมพ์ชื่อลงในช่อง **Name Box** ซึ่งอยู่ด้านซ้ายสุดของ Formula Bar หรือจะใช้เมนู **Formulas > Define Name** ตั้งชื่อแทนการใช้ Name Box ก็ได้

สมมติว่า ต้องการตั้งชื่อเซลล์ A1 ว่า Target ให้คลิกที่เซลล์ A1 ก่อนจากนั้นพิมพ์คำว่า Target ลงไปในช่อง Name Box แล้วกด Enter รับชื่อลงไป



หรือใช้วิธีตั้งชื่อผ่านเมนู **Formulas > Define Name**



หลักการตั้งชื่อ Range Name ที่ดี ให้ใช้ตัวอักษรภาษาอังกฤษตัวใหญ่ผสมตัวเล็ก อย่าใช้ตัวใหญ่ทั้งหมดหรือตัวเล็กทั้งหมด ถ้าอยากจะแทรกตัวเลขลงไปในเรื่องก็ได้ เพียงแต่ต้องใช้ตัวอักษรเป็นตัวแรกในชื่อก่อน ส่วนชื่อที่ตั้งนั้นควรตั้งให้สื่อถึงความหมายตรงกับเรื่องที่เราต้องการ และพยายามอย่าตั้งชื่อที่อาจซ้ำกับชื่อที่ใช้เรียกภายใน Excel หรือ VBE เช่น Sheet, Cell, Range, AB123 เป็นต้น

วิธีนำชื่อ Range Name มาใช้ใน VBE

ถ้าใช้ Macro Recorder ให้บันทึกเริ่มจากคลิกเข้าไปหาแฟ้มที่ต้องการให้ได้ก่อน จากนั้นให้กดปุ่ม **F5** แล้วดับเบิลคลิกชื่อที่ต้องการ จะได้รหัส VBA

```
Windows("Book1.xls").Activate
Application.Goto Reference:="Target"
ActiveCell.FormulaR1C1 = "123"
```

ถ้าเขียนรหัสเอง ให้ใช้ Target แทนลงไปในส่วนของชื่อชีทและตำแหน่งเซลล์ ดังนี้

```
Windows("Book1.xls").Activate
Range("Target") = 123
```

หมายเหตุ แม้จะใช้ Range Name ช่วยทำให้รหัส VBA ยึดหยุ่นตามการเปลี่ยนแปลงใน Excel ได้แล้วก็ตาม แต่ยังคงจำเป็นต้องใช้คำสั่ง Windows("Book1.xls").Activate

พาไปยังแฟ้มที่มี Range Name ชื่อ Target ก่อนเสมอ หรือถ้าไม่ต้องการใส่บรรทัดนี้ เราต้องเตรียมให้แฟ้ม Book1.xls เป็นแฟ้มที่กำลังถูกเลือกใช้งานไว้ก่อน จากนั้นจึงจะใช้คำสั่งเพียง Range("Target") = 123 เพราะ VBE จะตามหา Range Name ชื่อ Target ได้จากแฟ้มที่มีชื่อ Target ตั้งไว้แล้วเท่านั้น

วิธีย่อรหัส VBA ที่ใช้ชื่อ Range Name ให้สั้นลงโดยใช้ []

แทนที่จะต้องเสียเวลาพิมพ์ตำแหน่งเซลล์ที่ต้องการว่า Range("Target") เราสามารถใช้วิธี Evaluate ผ่านเครื่องหมาย [] ทำให้รหัสสั้นลงเหลือเพียง [Target] ดังนั้นหากเรากำลังเปิดแฟ้มที่มี Range Name ชื่อ Target อยู่แล้ว และต้องการส่งตัวเลข 123 ไปที่เซลล์ชื่อ Target จึงใช้รหัส VBA ที่เขียนสั้นๆง่ายๆ ดังนี้

```
[Target] = 123
```

การ Evaluate เป็นการสั่งให้ตัว VBE ประเมินว่า คำว่า Target คืออะไร ซึ่งเนื่องจากแฟ้มที่กำลังใช้งานอยู่มี Range Name ชื่อ Target ดังนั้น VBE จะทราบเองต่อไปว่าให้ส่งเลข 123 ไปที่ Target ซึ่งพบแล้วว่าเป็น Range Name คือตำแหน่งเซลล์ที่ตั้งชื่อไว้ว่า Target นั่นเอง

การ Evaluate ทำให้ VBE ทำงานช้าลงบ้าง แต่ทำให้เราเขียนรหัสอ้างถึงตำแหน่งอ้างอิงได้ง่ายกว่าเดิมมาก

VBA กับการจัดการฐานข้อมูล

ก่อนจะเร่งรีบหาทางใช้ VBA มาจัดการกับฐานข้อมูล ยังมีคำสั่งบนเมนูหรือสูตรสำเร็จรูปของ Excel ที่นำมาช่วยงานจัดการฐานข้อมูลได้อีก และเชื่อหรือไม่ว่า ลักษณะโครงสร้างตารางที่ใช้เก็บข้อมูลนั้นแหละเป็นสิ่งสำคัญที่สุด ถ้าโครงสร้างตารางจัดไว้ไม่ดี ออกแบบตารางไว้ผิด อย่างนี้เลยว่าจะใช้คำสั่งบนเมนู สูตร หรือแม้แต่ VBA มาช่วยในงานจัดการฐานข้อมูลได้ง่ายๆ

โครงสร้างตารางสำหรับเก็บข้อมูล

	A	B	C	D	E
1					
2		Id	Name	Amount	
3		a001	a	10	
4		a002	b	20	
5		a003	c	30	
6		a004	d	40	
7		a005	e	50	
8					
9					
10					

ตารางฐานข้อมูลที่ดีต้องมีลักษณะโครงสร้าง ตามกฎ 3 ข้อดังนี้

1. **หัวตารางต้องใช้พื้นที่เซลล์ 1 row เท่านั้น** จากภาพคือพื้นที่เซลล์ B2, C2, D2 แต่ถ้าอยากทำให้หัวตารางมีคำอธิบายเพิ่มเติมในอีกบรรทัด ให้ใช้วิธีกดปุ่ม **Alt+Enter** เพื่อขึ้นบรรทัดใหม่ในเซลล์เดิมต่อท้ายข้อความเดิม
2. **ห้ามเว้นว่าง row ทั้ง row ของทั้งรายการ** เช่น ห้ามเว้นเซลล์ B3:D3 หรือ B5:D5 เป็นต้น และไม่ควรเว้นว่างเซลล์ใดเซลล์หนึ่งโดยไม่จำเป็น โดยเฉพาะเซลล์ที่เก็บข้อมูลที่เป็นรหัสหรือชื่อสินค้า ซึ่งใช้เป็นข้อมูลสำหรับใช้ค้นหาข้อมูลอื่นในแนวรายการเดียวกัน
3. **ห้ามนำตารางเก็บข้อมูลไปติดกับเซลล์ข้อมูลอื่น** ทั้งด้านบน ด้านล่าง ด้านซ้าย และด้านขวาของตารางฐานข้อมูล ต้องเว้นว่างไว้รอบข้างอย่าง

น้อย 1 เซลล์โดยตลอด เพื่อให้ Excel รู้จักว่าพื้นที่เซลล์ที่ติดต่อกันนี้เป็นตารางเดียวกัน

ต่อเมื่อออกแบบตารางฐานข้อมูลไว้ถูกต้องแล้ว จะพบว่าเราสามารถใช้คำสั่งบนเมนู Data ต่อไปได้ทุกเมนู และใช้งานง่ายขึ้น โดยไม่จำเป็นต้องเสียเวลาเลือกพื้นที่ตารางทั้งหมดก่อนอีกต่อไป

ให้เริ่มจากคลิกเซลล์ใดเซลล์หนึ่งในพื้นที่ตารางเก็บข้อมูล แล้วเมื่อคลิกเลือกคำสั่งบนเมนู Data จะพบว่า Excel จะเลือกพื้นที่เฉพาะส่วนที่ต้องการนำมาใช้งานกับเมนูนั้นๆ ต่อให้เอง เช่น ถ้าสั่ง **Data > Sort** จะเลือกพื้นที่ B3:D7 เฉพาะส่วนที่เป็นข้อมูลรายการ หรือถ้าสั่ง **Data > Filter** จะเกิดปุ่ม Filter ขึ้นเฉพาะหัวตารางด้านบน B2:D2

สูตรสำเร็จรูปที่ใช้จัดการฐานข้อมูล

หลังจากจัดเก็บข้อมูลไว้ในตารางที่มีโครงสร้างถูกต้องตามที่กำหนดไว้ข้างต้นแล้ว เราสามารถเลือกใช้สูตรต่อไปนี้เพื่อค้นหาข้อมูลหรือคำตอบที่ต้องการต่อไป

โครงสร้างสูตร IF ใช้หาคำตอบตามเงื่อนไข

=IF(เงื่อนไข, ผลหากเงื่อนไขเป็นจริง, ผลหากเงื่อนไขเป็นเท็จ)

ตัวอย่าง ให้หาคำตอบถ้ายอดรวมของ Amount ทั้งหมด มีค่ารวมแล้วมีค่ามากกว่าหรือเท่ากับ 100 ให้ตอบว่า OK แต่ถ้ายอดรวมไม่ถึง 100 ให้ตอบว่า No จะต้องสร้างสูตรต่อไปนี้

=IF(Sum(D3:D7)>=100, "OK", "No")

คำตอบ คือ OK

โครงสร้างสูตร VLookup ใช้หาค่าในตาราง

=VLookup(รหัสที่ค้นหา, ตารางเก็บข้อมูล, เลข column ของคำตอบที่ต้องการ, 0)

ตัวอย่าง ต้องการหาว่า รหัส Id a003 มีชื่ออะไร และมียอดเป็นเท่าใด ให้ใช้สูตรต่อไปนีตามลำดับ

=VLookup("a003", B3:D7 , 2, 0)

คำตอบ คือ C

=VLookup("a003", B3:D7 , 3, 0)

คำตอบ คือ 30

เลข 2 และ 3 ที่ใส่ไว้ในสูตร คือ เลขของ column ของ Name และ Amount ซึ่งอยู่ใน column ที่ 2 และ 3 ตามลำดับในพื้นที่ตาราง B2:D7

เลข 0 ที่ใส่ไว้ด้านหลังสุดในสูตร เป็นการกำหนดให้ใช้สูตร VLookup หาค่าแบบ Exact Match โดยจะหาค่าเฉพาะเมื่อมีค่าตรงกับรหัสที่ค้นหาเท่านั้น แต่ถ้าไม่ใส่เลข 0 จะทำให้ VLookup ทำงานแบบ Approaching Match โดยจะหาค่ามาให้เสมอ แม้ว่าไม่มีรหัสตรงกันก็ตาม ซึ่งโดยทั่วไปถ้าใช้รหัสเป็นค่าที่ใช้ค้นหาคำตอบ ต้องใช้แบบ Exact Match

นอกจากนั้น เลข 0 มีความหมายเท่ากับ False แปลว่า ไม่ เพื่อระบุว่า ตารางที่ใช้ค้นหาตำแหน่งนั้น ไม่ต้องเรียงลำดับก็ได้ แต่ถ้าไม่ใส่เลข 0 จะต้องเรียงรหัสที่เก็บไว้จากน้อยไปมากก่อนจึงจะหาคำตอบได้ถูกต้อง

โครงสร้างสูตร Match ใช้หาตำแหน่งของค่าในตาราง

=Match(รหัสที่ค้นหา, พื้นที่แนวดิ่งของรหัสทั้งหมด, 0)

ตัวอย่าง ต้องการหาว่ารหัส a003 เป็นรายการที่เท่าใด ให้ใช้สูตรดังนี้

=Match("a003", B3:B7, 0)

คำตอบ คือ 3

นอกจากพื้นที่แนวดิ่งแล้ว ยังใช้กับพื้นที่แนวนอนก็ได้ และโดยทั่วไปให้ใส่เลข 0 ต่อท้ายด้านหลังสุดในสูตรเสมอ ซึ่งเลข 0 มีความหมายเท่ากับ False แปลว่า ไม่ เพื่อระบุว่า ตารางที่ใช้ค้นหาตำแหน่งนั้น ไม่ต้องเรียงลำดับก็ได้

โครงสร้างสูตร Index ใช้หาค่า ที่อยู่ตามตำแหน่งของค่าในตาราง

=Index(ตารางเก็บข้อมูล, เลขที่ Row, เลขที่ Column)

ตัวอย่าง ต้องการหาค่าจากพื้นที่ตาราง B3:D7 ณ ตำแหน่ง row ที่ 2 ตัดกับ column ที่ 3 ให้ใช้สูตรดังนี้

=Index(B3:D7, 2, 3)

คำตอบ คือ 20

โครงสร้างสูตร Offset แบบสั้น ใช้หาค่า ที่อยู่ตามตำแหน่งถัดไปจากตำแหน่งเซลล์อ้างอิงที่กำหนด

=Offset(เซลล์อ้างอิง, จำนวน row ถัดไป, จำนวน column ถัดไป)

ตัวอย่าง ต้องการหาค่า ที่อยู่ถัดจากเซลล์ B3 ลงไป 1 row และถัดไปจากเซลล์ B3 ไปด้านขวา 2 column ให้ใช้สูตรดังนี้

=Offset(B3, 1, 2)

คำตอบ คือ 20

โครงสร้างสูตร Offset แบบเต็ม ใช้หาค่าหลายค่า เสมือนเป็นพื้นที่ตาราง

```
=Offset(เซลล์อ้างอิง, จำนวน row ถัดไป, จำนวน column ถัดไป,  
ความสูงของตาราง, ความกว้างของตาราง)
```

ตัวอย่าง ต้องการใช้สูตรกำหนดพื้นที่ของตัวเลข Amount ซึ่งจะเห็นได้ว่ามีเลข 10, 20, 30, 40, 50 เก็บไว้ ให้ใช้สูตรดังนี้

```
=Offset(B3, 0, 2, 5, 1)  
คำตอบ คือ {10;20;30;40;50}
```

สาเหตุที่ใส่เลข 0,2,5,1 นั้น เนื่องจากตารางข้อมูล Amount D3:D7 มีตำแหน่ง D3 อยู่ในแนว row เดียวกันกับ B3 จึงมีจำนวน row ถัดไปเท่ากับ **0** และตำแหน่ง D3 อยู่ถัดไปจาก B3 อีก **2** column โดย D3:D7 มีพื้นที่ความสูง **5** row และกว้าง **1** column

สูตรที่สร้างขึ้นนี้ อาจไม่เห็นคำตอบครบตามต้องการ ขอให้กดปุ่ม F2 แล้วตามด้วย F9 จะพบตัวเลข {10;20;30;40;50} และสังเกตว่า ระหว่างตัวเลขนั้น มีเครื่องหมาย semi-colon ; คั่น ซึ่งเครื่องหมาย ; นี้แสดงว่า ตัวเลขเป็นแนวตั้ง (ถ้าคั่นด้วยเครื่องหมาย comma , จะแสดงว่าเป็นแนวนอน) เมื่อเห็นตัวเลขแล้ว ให้กดปุ่ม Esc เพื่อกลับไปเป็นสูตรตามเดิม

โดยทั่วไปจะไม่ค่อยพบสูตร Offset แบบเต็มใช้ในเซลล์ใด แต่จะนำสูตร Offset แบบเต็ม ไปสร้างต่อเป็น Formula Name ผ่านเมนู **Formula > Define Name** เพื่อตั้งเป็นชื่อ Formula Name นำมาใช้แทน Range Name

Offset ทำหน้าที่กำหนดขนาดตารางได้เอง (Dynamic Range)

สูตรช่วยในการกำหนดพื้นที่ตาราง ที่ขยายหรือหดได้ตามจำนวนข้อมูล เรียกว่า

Dynamic Range เช่น หากต้องการทำให้พื้นที่ของ Amount ซึ่งเดิมมาจากเซลล์

D3:D7 นั้น สามารถขยายหรือหดพื้นที่ได้ตามข้อมูลที่เพิ่มหรือลดได้เอง ให้สั่ง

Formula > Define Name ตั้งชื่อ Formula Name ว่า Amount และกำหนดสูตรต่อไปนีลงในช่อง Refers to

```
=Offset($B$3, 0, 2, CountA($D:$D)-1, 1)
```

สังเกตว่า ตำแหน่งของเซลล์ที่อ้างอิงในสูตรเป็น Absolute Reference โดยมี

เครื่องหมาย \$ กำกับตำแหน่งทั้ง row และ column ทั้งนี้เป็นการสร้างตามหลักของ

Formula Name เพื่อให้ Amount มีตำแหน่งถูกต้องและมีตำแหน่งแน่นอน

สูตร CountA(\$D:\$D)-1 ทำหน้าที่คำนวณหาความสูง โดยนับจำนวนรายการข้อมูล

ทั้งหมดจาก column D แต่เนื่องจากสูตร CountA นับหัวตารางเซลล์ D2 เกินเข้ามา 1

เซลล์ จึงต้องลบจำนวนที่นับได้ทั้งไป 1 เซลล์

โครงสร้างสูตร CountIF

ใช้นับจำนวนเซลล์ เท่าที่มีค่าตามเงื่อนไข

```
=CountIF(พื้นที่ตาราง, "เงื่อนไข")
```

ตัวอย่าง ต้องการนับจำนวนรหัส a003 ว่ามีกี่เซลล์ ให้ใช้สูตรดังนี้

```
=CountIF(B3:B7, "a003")
```

คำตอบ คือ 1

หลักการใช้ VBA จัดการฐานข้อมูล

1. อย่าใช้ VBA ถ้าเราสามารถใช้เมนูหรือสูตรหาคำตอบที่ต้องการได้อยู่แล้ว นอกจากช่วยประหยัดเวลาของเรา ไม่ต้องหาทางสร้างรหัส VBA ขึ้นเอง การใช้เมนูหรือสูตร ยังทำให้ Excel ทำงานเร็วกว่าการใช้ VBA อย่างมาก (คำสั่งบนเมนูและสูตรที่ Microsoft สร้างไว้ให้ นั้น จะทำงานได้เร็วมาก เพราะไม่ต้องเสียเวลาแปลรหัสเป็นภาษาเครื่อง ต่างจาก VBA ที่เราเขียนเอง ซึ่งยากจะเขียนรหัสที่มีประสิทธิภาพได้เทียบเท่า Microsoft)
2. ต้องออกแบบตารางให้ถูกต้อง แล้วทดลองใช้คำสั่งบนเมนู และใช้สูตร จนได้ขั้นตอนที่ลัดที่สุดที่ทำได้
3. ใช้ Macro Recorder บันทึกขั้นตอนการใช้คำสั่งบนเมนู แทนการเขียนรหัสเองทั้งหมด
4. รหัส VBA ที่สร้างไว้นั้น ต้องสร้างแบบยืดหยุ่น สามารถใช้รหัสเดิมทำงานต่อไปได้ตลอด ไม่ต้องเสียเวลาย้อนกลับมาแก้ไขรหัสอีกในภายหลัง ไม่ว่าโครงสร้างตาราง ชื่อชีท ชื่อแฟ้ม ชื่อโฟลเดอร์จะต่างไปจากเดิมหรือไม่อย่างไร
5. ค่าคงที่หรือตัวแปรทั้งหมดที่ใช้ในรหัส VBA ให้ link ต่อมาจาก Excel โดยใช้ Range Name หรือ Formula Name เป็นสื่อกลาง
6. ถ้าจำเป็นต้องแก้ไข ให้แก้ไขเฉพาะส่วนที่อยู่ใน Excel เนื่องจากเป็นสิ่งที่เราคุ้นเคย ใช้งานทุกวัน สามารถแก้ไขได้ง่ายและสะดวกกว่าเข้าไปแก้ไขรหัสใน VBE
7. หลีกเลี่ยงการลบรายการที่เลิกใช้งานทิ้ง เพราะการลบข้อมูลทิ้ง ย่อมเสียข้อมูลเก่านั้นไป แต่ให้ใช้วิธีเขียนกำกับรายการที่เลิกใช้งาน เช่น ใช้เซลล์ที่มีเลข 99 กำกับรายการใดเพื่อแสดงว่า รายการนั้นเลิกใช้ไปแล้ว
8. หลีกเลี่ยงการแทรกรายการใหม่เข้าไประหว่างรายการเก่า แต่ให้บันทึกข้อมูลรายการใหม่ ต่อท้ายรายการเก่าทั้งหมดต่อกันไปเรื่อยๆ แล้วจึงนำข้อมูลไปจัดเรียงในพื้นที่อื่น เป็นตารางใหม่ที่จัดเรียงตามต้องการ
9. ควรเลือกใช้ VBA ลดขั้นตอนที่จำเป็นเท่านั้น อย่าพยายามเขียน VBA เพื่อหาทางทำให้งานทุกขั้นตอนทำงานเองโดยอัตโนมัติ เพราะรหัสจะมีความ

ซับซ้อนและยากขึ้นมาก ลองนึกเผื่อไว้ด้วยว่า คนอื่นรุ่นหลังจะสามารถ
แกะรหัสเดิม มาแก้ไขหรือพัฒนาต่อไหวหรือไม่

วิธีใช้รหัส VBA แบบ Evaluate ในการรับส่งข้อมูล

เดิมวิธีที่ใช้ในการรับส่งข้อมูลใน Excel เป็นการใช้สูตร link ข้อมูลจากเซลล์ต้นทางมาใช้ต่อที่เซลล์ปลายทาง ถ้าตารางข้อมูลมีขนาดตายตัว จะใช้วิธีสร้างสูตร link เตรียมไว้ตั้งแต่ต้นให้รับส่งข้อมูลเซลล์ต่อเซลล์ แต่ถ้าตารางมีจำนวนรายการข้อมูลที่ไม่แน่นอน จะต้องเป็นหน้าที่ของเราที่ต้องคอยสร้างสูตรรับข้อมูลตามจำนวนเซลล์ต้นทาง หรือบางคนอาจใช้วิธีสร้างสูตรเพื่อไว้เติมตารางปลายทางไว้ก่อน โดยไม่สนใจว่าตารางต้นทางจะมีข้อมูลมากน้อยเพียงใด ยอมให้เซลล์ใดยังไม่มีข้อมูลส่งมา จะแสดงเลข 0 ไปก่อน

เนื้อหาในบทนี้จะหาทางใช้ VBA ช่วยในการรับส่งข้อมูลแทนการใช้สูตร จะได้ไม่ต้องเสียแรงเสียเวลาวิ่งวนกลับมาสร้างสูตร link และไม่ต้องสร้างสูตร link เตรียมไว้ก่อนเติมตาราง ซึ่งจะช่วยให้แฟ้มที่มีตารางสูตรหลายๆเซลล์ กลายเป็นแฟ้มที่มีขนาดแฟ้มใหญ่โดยไม่จำเป็น

ชุดคำสั่งที่ใช้รับส่งข้อมูล มีเพียง 2 บรรทัด

```
Sub SendData()  
    MyVar = [Source]  
    [Target] = MyVar  
End Sub
```

ข้อกำหนดในการใช้งาน

1. Source และ Target เป็น Range Name หรือ Formula Name ก็ได้
2. Source คือ ตารางต้นทางที่ส่งข้อมูล
3. Target คือ ตารางปลายทางที่รับข้อมูล
4. ขนาดตารางของทั้ง Source และ Target ต้องมีขนาดเท่ากัน เช่น ถ้า Source เป็นเซลล์เดียว ดังนั้น Target ก็ต้องเป็นเซลล์เดียวด้วย แต่ถ้า Source เป็นตารางที่มีขนาดความสูง 2 row ความกว้าง 3 column ดังนั้น Target ต้องเป็นตารางที่มีขนาดความสูง 2 row ความกว้าง 3 column เช่นกัน

5. ขณะที่สั่งให้คำสั่งชุดนี้ทำงาน แฟ้มที่มีตารางที่ตั้งชื่อว่า Source และ Target ต้องเปิดและกำลังใช้งานอยู่
6. Source หรือ Target ไม่จำเป็นต้องเป็นตารางในชีทเดียวกัน โดยจะเป็นตารางที่อยู่ต่างชีทหรือต่างแฟ้มกันก็ได้ เช่น Target เป็นชื่อ Range Name ที่ตั้งไว้ในแฟ้ม Book1.xls แต่อาจกำหนดให้ Refers to พื้นที่ตารางใน Book2.xls (ในกรณีนี้ ขณะที่สั่งให้ชุดคำสั่งนี้ทำงาน ต้องเปิดแฟ้ม Book1.xls และ Book2.xls และต้องเลือกอยู่ในแฟ้ม Book1.xls เพราะเป็นแฟ้มที่มีชื่อ Range Name Source และ Target)
7. สามารถตั้งชื่อตารางเป็นชื่ออื่นได้ตามต้องการ โดยแก้ไขชื่อที่ใช้ในรหัสให้ถูกต้องตามชื่อที่ตั้งไว้ ต้องสะกดชื่อตามให้ถูกต้องตรงกันทุกตัวอักษร ชื่อในรหัสไม่จำเป็นต้องใช้อักษรตัวใหญ่ตัวเล็กตามชื่อใน Excel แต่ถ้าตรงกันได้ทุกตัวจะดีกว่า

คำอธิบายขั้นตอนการทำงานของชุดคำสั่ง SendData

1. MyVar เป็นชื่อตัวแปรที่ตั้งขึ้น (Array Variable) ซึ่งจะเปลี่ยนเป็นชื่ออื่นใดก็ได้ สำหรับรับข้อมูลที่มาจาก Range Name ชื่อ Source
2. จากนั้น Range Name ชื่อ Target จะรับข้อมูลที่เก็บไว้ใน MyVar
3. คำว่า Source และ Target ที่พิมพ์ไว้ในระหว่างเครื่องหมาย [] เป็นการใส่แบบ Evaluate ทำให้ VBE ค้นหาตัวเองว่าทั้ง Source และ Target เป็น Range Name
4. เราใช้ MyVar เป็นสื่อกลางในการรับส่งข้อมูลระหว่าง Source และ Target ซึ่งเฉพาะตัวค่าของข้อมูลเท่านั้นที่จะถูกส่งไปเก็บไว้ที่ Target ถ้าใน Source มีเซลล์สูตร $=1+2$ จะส่งผลลัพธ์เป็นเลข 3 ไปเก็บไว้ที่ Target สิ่งอื่นนอกจากค่าผลลัพธ์ เช่น Format จะไม่ถูกส่งออกไป
5. ถ้าตัด MyVar ออกแล้วใช้คำสั่ง [Target] = [Source] บรรทัดเดียว จะใช้รับส่งข้อมูลได้เฉพาะเซลล์เดียวเท่านั้น
6. กรณีต้องการสั่งให้รหัสชุดนี้ทำงาน และในขณะนั้นมีแฟ้มหลายแฟ้มกำลังเปิดใช้งานร่วมกันอยู่ ควรเพิ่มรหัสบรรทัดแรก ให้ทำหน้าที่ย้ายไปดูแฟ้มที่

มี Range Name ชื่อ Source และ Target ตั้งชื่อไว้ เช่น ถ้าเป็น Range Name ที่อยู่ในแฟ้มชื่อ Book1.xls ให้แก้ไขรหัสเป็น

```
Sub SendData()
    Windows("Book1.xls").Activate
    MyVar = [Source]
    [Target] = MyVar
End Sub

หรือ

Sub SendData()
    ThisWorkbook.Activate
    MyVar = [Source]
    [Target] = MyVar
End Sub
```

หมายเหตุ ThisWorkbook หมายถึง ตัวแฟ้มที่มี Module เก็บชุดคำสั่งที่กำลังทำงาน ดังนั้นเมื่อชุดคำสั่ง SendData ทำงาน แล้วพบคำสั่ง ThisWorkbook.Activate จะทำให้แฟ้มที่เก็บชุดคำสั่ง SendData ถูกเลือกขึ้น ซึ่งย่อมเป็นแฟ้มเดียวกันกับแฟ้มที่มี Range Name ชื่อ Source และ Target นั้นเอง

กรณีใช้ Macro Recorder ในการรับส่งข้อมูล

สมมติว่าเรากำลังอยู่ในแฟ้มที่มี Range Name ชื่อ Source และ Target และต้องการสร้างชุดคำสั่งเพื่อทำหน้าที่ส่งค่ารวมทั้ง Format จาก Source ไปยัง Target ให้ใช้ Recorder บันทึกตามขั้นตอนต่อไปนี้

1. กดปุ่ม F5 เลือกไปยัง Source
2. กดปุ่ม Ctrl+c เพื่อส่ง Copy
3. กดปุ่ม F5 เลือกไปยัง Target
4. กดปุ่ม Enter เพื่อ Paste

ชุดคำสั่งที่ได้จากการใช้ Macro Recorder

```
Sub CopySource2Target()
    Application.Goto Reference:="Source"
    Selection.Copy
    Application.Goto Reference:="Target"
    ActiveSheet.Paste
End Sub
```

แต่ถ้าไม่ต้องการนำ Format ตามไปที่ Target ด้วย ให้เปลี่ยนการบันทึก Macro ขั้นที่ 4 เป็น การคลิกขวาแล้วเลือก Paste Special > Values จะทำให้เกิดชุดคำสั่งต่างจากเดิมเล็กน้อยเป็น

```
Sub CopyDataSource2Target()
    Application.Goto Reference:="Source"
    Selection.Copy
    Application.Goto Reference:="Target"
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone,
    SkipBlanks:=False, Transpose:=False
End Sub
```

ซึ่งจะพบวาร์หัสในขั้นตอนการ Paste Special ที่ได้จากการบันทึก Macro เป็นรหัสที่ยาวเกินต้องการ ขอให้ตัดเหลือเพียง Selection.PasteSpecial Paste:=xlPasteValues ก็พอ กลายเป็นชุดคำสั่งที่สั้นและอ่านได้ความตามที่ต้องการมากขึ้นเป็น

```
Sub CopyDataSource2Target()
    Application.Goto Reference:="Source"
    Selection.Copy
    Application.Goto Reference:="Target"
    Selection.PasteSpecial Paste:=xlPasteValues
End Sub
```

วิธีส่งข้อมูลไปยังเซลล์ภายในตารางที่กำหนด


หัวใจของเคล็ดลับการเพิ่มผลงาน ลดความซับซ้อนของงานด้วย Excel VBA มีได้อยู่ที่การหาทางทำให้สามารถทำงานเสร็จเร็วขึ้น และทำให้งานที่มีขั้นตอนซับซ้อน กลายเป็นงานง่ายขึ้นเท่านั้น ยังต้องหาทางทำให้ตัวรหัส VBA ที่ใช้ มีความซับซ้อนลดลงไปด้วย ซึ่งในบทที่แล้วได้แนะนำชุดคำสั่งชื่อ SendData และจากนี้ไปเราจะมาดูกันว่า รหัสคำสั่งเพียงแค่ 2 บรรทัดใน SendData จะนำไปใช้ในการรับส่งข้อมูลในแบบต่างๆกันได้อย่างไร โดยมีข้อแม้สำคัญว่า เราจะไม่แก้ไขรหัส VBA 2 บรรทัดนั้นอีกเลย

ก่อนอื่นมาดูกันอีกทีว่า ชุดคำสั่ง SendData มีหน้าตาอย่างไรและทำงานได้อย่างไร

```
Sub SendData()  
    MyVar = [Source]  
    [Target] = MyVar  
End Sub
```

ชุดคำสั่งนี้ สั่งให้ข้อมูลที่อยู่ใน Range Name ชื่อ Source ส่งไปปักไว้ในตัวแปรชื่อ MyVar จากนั้นจึงสั่งให้ MyVar ส่งข้อมูลที่เก็บไว้ต่อไปยัง Range Name ชื่อ Target

ตัวอย่าง

	A	B	C	D	E	F
1				MyData		
2	Row	5				
3	Col	2				
4	Input	777				
5						
6					777	
7						
8						
9						
10						
11						

สิ่งที่ต้องการ

ส่งตัวเลข 777 หรือค่าใดๆที่บันทึกไว้ในเซลล์ B4 ไปเก็บไว้ในตารางชื่อ MyData ณ ตำแหน่ง row ที่ 5 ดัดกับ column ที่ 2 หรือตำแหน่งอื่นใดก็ได้ที่อยู่ใน MyData

ขั้นตอนการสร้างงาน

1. สร้าง Range Name ชื่อ Source ให้กับเซลล์ B4
2. สร้าง Range Name ชื่อ MyData ให้กับเซลล์ D2:F10
3. สร้าง Formula Name ชื่อ Target ให้เป็นสูตร
=INDEX(MyData, \$B\$2, \$B\$3)

วิเคราะห์สูตร

```
Target
=INDEX( MyData, $B$2, $B$3 )
```

เมื่อกำหนดตำแหน่ง Row และ Column ลงไปในเซลล์ B2 และ B3 ตามลำดับ จะส่งผลให้ข้อมูลจาก Source ส่งไปที่ MyVar ต่อไปยัง Target แล้วสูตร Index จะเป็นตัวกำหนดตำแหน่ง Row และ Column ของเซลล์ที่รับค่าในที่สุด

ต้องระวังในการกำหนดตำแหน่ง Row และ Column เพราะสูตร Index สามารถรับค่าลงไปเฉพาะขอบเขตพื้นที่ของ MyData เท่านั้น ซึ่ง MyData มีขนาดความสูง 9 row และมีความกว้าง 3 column จึงทำให้เลข Row และ Column ช่วงที่ใช้ได้ คือ Row ที่ 1 - 9 และ Column ที่ 1 - 3

วิธีส่งข้อมูลไปยังเซลล์รับข้อมูล ณ ตำแหน่งใดก็ได้

ตัวอย่างนี้คล้ายกับตัวอย่างที่ใช้สูตร Index แต่เลือกใช้สูตร Offset แทน เพื่อให้ส่งค่าไปยังเซลล์ใดก็ได้ ไม่ต้องจำกัดขอบเขต

ชุดคำสั่ง : SendData

	A	B	C	D	E	F	G
1				Ref			
2	Row	5					
3	Col	2					
4	Input	777					
5							
6						777	
7							
8							
9							

สิ่งที่ต้องการ

ส่งตัวเลข 777 หรือค่าใดๆที่บันทึกไว้ในเซลล์ B4 ไปเก็บไว้ในตาราง โดยใช้ตำแหน่งอ้างอิงจากเซลล์ D1 ว่า เซลล์ที่ต้องการรับข้อมูล อยู่ห่างจากเซลล์ D1 เป็นระยะทาง 5 row และห่าง 2 column

ขั้นตอนการสร้างงาน

1. สร้าง Range Name ชื่อ Source ให้กับเซลล์ B4
2. สร้าง Range Name ชื่อ Ref ให้กับเซลล์ D1
3. สร้าง Formula Name ชื่อ Target ให้เป็นสูตร
=OFFSET(Ref, \$B\$2, \$B\$3)

วิเคราะห์สูตร

Target
=OFFSET(Ref, \$B\$2, \$B\$3)

สูตรนี้ทำงานคล้ายกับ Index เพียงแต่กำหนดตำแหน่ง Row และ Column ถัดไปจากตำแหน่งเซลล์ชื่อ Ref ซึ่งตำแหน่ง Row และ Column นี้เป็นได้ทั้งค่าบวก ลบ และ

ศูนย์ (ถ้าตำแหน่งอยู่เหนือเซลล์ Ref หรืออยู่ด้านซ้ายเซลล์ Ref ให้กำหนดเป็นเลขลบ)

เมื่อกำหนดตำแหน่ง Row และ Column ลงไปในเซลล์ B2 และ B3 ตามลำดับ จะส่งผลให้ข้อมูลจาก Source ส่งไปที่ MyVar ต่อไปยัง Target แล้วสูตร Offset จะเป็นตัวกำหนดตำแหน่ง Row และ Column ของเซลล์ที่รับค่าในที่สุด

ตัวอย่างนี้ แม้จะไม่จำกัดขอบเขตว่าต้องอยู่ภายในตารางที่กำหนด แต่อย่างไรก็ตามยังต้องอยู่ภายในชีท จึงต้องระวังตัวเลขตำแหน่งเซลล์ว่า อย่าใช้ตำแหน่งที่เป็นไปไม่ได้ เช่น กำหนดค่า B2 เป็นเลข -100 ซึ่งย้อนขึ้นไปด้านบนเหนือ row ที่ 1 เสียอีก

วิธีส่งข้อมูลจากตารางที่มีขนาดไม่แน่นอน

ตัวอย่างที่ผ่านมาเป็นการรับส่งข้อมูลเพียงเซลล์เดียว ซึ่งสามารถเลือกใช้สูตร Index หรือ Offset ก็ได้ แต่ถ้าต้องการรับส่งข้อมูลจากตารางที่มีขนาดตั้งแต่ 2 เซลล์ขึ้นไป ต้องเลือกใช้สูตร Offset แบบเต็ม เพราะสูตร Offset นี้เท่านั้นที่สามารถกำหนดส่วนของความสูงและความกว้างของตาราง

เนื่องจากตารางข้อมูลโดยทั่วไป มีความกว้างคงที่เท่ากับจำนวน Field ซึ่งแยกเป็น column เก็บข้อมูลแต่ละเรื่อง แต่ความสูงจะเพิ่มหรือลดตามจำนวนรายการ จึงทำให้เฉพาะความสูงของตารางเท่านั้นที่มีขนาดเปลี่ยนแปลง จึงขอให้สังเกตว่า เราจะใช้สูตร CountA เพื่อคำนวณหาความสูงให้กับสูตร Offset

ชุดคำสั่ง : SendData

	A	B	C	D	E	F	G	H	I
1									
2	Source	Id	Name	Amount		Target	Id	Name	Amount
3	Ref1	a001	a	10		Ref2	a001	a	10
4		a002	b	20			a002	b	20
5		a003	c	30			a003	c	30
6		a004	d	40			a004	d	40
7		a005	e	50			a005	e	50
8									
9									
10									
11									

สิ่งที่ต้องการ

ส่งข้อมูลเท่าที่มีอยู่จากตารางด้านซ้ายไปเก็บไว้ในตารางด้านขวา ซึ่งจำนวนรายการข้อมูลที่มีนั้นอาจเพิ่มหรือลดก็ได้

ขั้นตอนการสร้างงาน

1. สร้าง Range Name ชื่อ Ref1 ให้กับเซลล์ B3
2. สร้าง Range Name ชื่อ Ref2 ให้กับเซลล์ G3
3. สร้าง Formula Name ชื่อ Source ให้เป็นสูตร
=OFFSET(Ref1, 0, 0, COUNTA(\$B:\$B)-1, 3)
4. สร้าง Formula Name ชื่อ Target ให้เป็นสูตร
=OFFSET(Ref2, 0, 0, COUNTA(\$B:\$B)-1, 3)

วิเคราะห์สูตร

```
Source
=OFFSET( Ref1, 0, 0, COUNTA($B:$B)-1, 3 )
Target
=OFFSET( Ref2, 0, 0, COUNTA($B:$B)-1, 3 )
```

เนื่องจากตำแหน่งเซลล์หัวมุมของตารางทั้งคู่ อยู่ตรงกับตำแหน่งของ Ref1 และ Ref2 ดังนั้นจึงกำหนดตำแหน่ง row และ column ถัดไป เท่ากับ 0

สูตรทั้งคู่กำหนดค่าความสูงเท่ากับ COUNTA(\$B:\$B)-1 โดยใช้ข้อมูลใน column ของ Id เป็นหลักในการนับ แต่เนื่องจากมีเซลล์คำว่า Id อยู่ใน column นี้ด้วย จึงต้องลบ 1 ออกจาก COUNTA(\$B:\$B) ส่วนความกว้างของตาราง กำหนดให้มีขนาดคงที่เท่ากับ 3 column

ขอให้สังเกตว่า ชุดคำสั่ง SendData จะส่งข้อมูลจากตารางด้านซ้าย ไปทับข้อมูลเดิมที่มีอยู่แล้วในตารางด้านขวา ซึ่งเหมาะกับกรณีที่ตารางซ้ายมีจำนวนรายการเพิ่มขึ้นเรื่อยๆ เท่านั้น ไม่เหมาะกับกรณีที่ตารางซ้ายมีจำนวนรายการน้อยกว่าตารางขวา หากต้องการปรับปรุงชุดคำสั่งให้ทำงานได้ถูกต้อง ควรเพิ่มคำสั่งลบข้อมูลเดิมที่มีในตารางขวาทั้งหมดก่อน แล้วจึงสั่งให้ส่งข้อมูลไปเก็บ

นอกจากนั้นข้อมูล Id ต้องบันทึกในเซลล์ติดต่อกันไป (ห้ามเว้นเซลล์ว่างไว้) และห้ามบันทึกค่าอื่นในเซลล์ใดๆบน column B ที่ไม่เกี่ยวข้องกับรหัส เพราะจะทำให้สูตร CountA นับจำนวนรายการผิด ส่งผลให้ค่าความสูงที่ส่งต่อไปใช้ในสูตร Offset ผิดพลาดตามไปด้วย

วิธีส่งข้อมูลไปบันทึกต่อท้ายรายการที่มีอยู่แล้ว

ตัวอย่างนี้จะช่วยแก้ปัญหาให้กับงานได้มากมายหลายประเภท ซึ่งต้องการกรอกข้อมูลลงไปในเซลล์ แล้วต้องการนำข้อมูลที่กรอกไว้ไปเก็บต่อท้ายข้อมูลที่มีอยู่แล้ว ทำให้เกิดเป็นตารางฐานข้อมูลในที่สุด

ชุดคำสั่ง : SendData

	A	B	C	D	E	F	G	H	I
1									
2		Id	Name	Amount		Target	Id	Name	Amount
3	Source	a004	d	40		Ref	a001	a	10
4							a002	b	20
5							a003	c	30
6									
7									
8									
9									
10									
11									

สิ่งที่ต้องการ

ส่งข้อมูลที่กรอกลงในเซลล์ B3:D3 ไปเก็บต่อท้ายรายการในตารางด้านขวา

ขั้นตอนการสร้างงาน

1. สร้าง Range Name ชื่อ Source ให้กับเซลล์ B3:D3
2. สร้าง Range Name ชื่อ Ref ให้กับเซลล์ G3
3. สร้าง Formula Name ชื่อ Target ให้เป็นสูตร
=OFFSET(Ref, COUNTA(\$G:\$G)-1, 0, 1, 3)

วิเคราะห์สูตร

Target
=OFFSET(Ref, COUNTA(\$G:\$G)-1, 0, 1, 3)

ขอให้สังเกตเลข 0, 1, 3 ที่อยู่ด้านท้ายของสูตร Offset ก่อน

- เลข 0 กำหนดตำแหน่งของข้อมูล ให้เริ่มในแนว column เดียวกันกับ Ref
- เลข 1 คือ ความสูงของรายการข้อมูล ซึ่งต้องมีความสูงคงที่ 1 row เสมอ

- เลข 3 คือ ความกว้างของรายการ ซึ่งประกอบด้วย Id, Name, Amount จึงกำหนดให้กว้าง 3 column คงที่

ส่วนของสูตร COUNTA(\$G:\$G)-1 เป็นตัวช่วยกำหนดตำแหน่งรายการใหม่ต่อท้ายรายการเดิม โดย COUNTA(\$G:\$G) จะนับจำนวนเซลล์ที่มีข้อมูลใน column G ทั้งหมด แต่เนื่องจากนับเซลล์คำว่า Id รวมเกินมา 1 เซลล์ จึงต้องลบ COUNTA(\$G:\$G) ออกเสีย 1 ตำแหน่ง ทั้งนี้เพื่อให้ตรงกับจำนวน row ที่ต้องนับให้ ถัดไปจากตำแหน่งของเซลล์ Ref

ดังนั้นสูตร Offset นี้ จึงทำหน้าที่ส่งรายการข้อมูลที่กรอกไว้ในตารางที่มีความสูง 1 row และกว้าง 3 column จาก Range Name ชื่อ Source ไปยัง Target ซึ่งมีขนาด ความสูงและความกว้างเดียวกัน เพียงแต่ Target จะขยับไปหาตำแหน่งรายการถัดไป จาก Ref ตามที่นับได้ด้วยสูตร CountA-1

วิธีส่งข้อมูลกลับไปแก้รายการเดิมที่มีอยู่แล้ว

แทนที่จะส่งข้อมูลใหม่ไปต่อท้ายรายการทั้งหมดที่มีอยู่เดิม คราวนี้จะหาทางส่งข้อมูลไปแก้รายการเก่า (ถ้ามี)

ชุดคำสั่ง : SendData

	A	B	C	D	E	F	G	H	I
1									
2		Id	Name	Amount		Target	Id	Name	Amount
3	Source	a002	boy	20		Ref	a001	a	10
4							a002	b	20
5							a003	c	30
6									
7									
8									
9									
10									
11									

สิ่งที่ต้องการ

ส่งข้อมูลที่กรอกลงในเซลล์ B3:D3 กลับไปแก้ไขรายการเก่า โดยใช้ค่าที่กรอกนั้น
ส่งไปทับรายการเดิมที่มีอยู่แล้วในตารางด้านขวา เช่น จากภาพ ให้ส่งรายการ a002 ที่
ใช้ชื่อใหม่ว่า boy ไปแก้ไขรายการ a002 เดิมซึ่งใช้ชื่อว่า b ให้เป็น boy

ขั้นตอนการสร้างงาน

1. สร้าง Range Name ชื่อ Source ให้กับเซลล์ B3:D3
2. สร้าง Range Name ชื่อ Ref ให้กับเซลล์ G3
3. สร้าง Range Name ชื่อ Id ให้กับเซลล์ G3:G10
4. สร้าง Formula Name ชื่อ Target ให้เป็นสูตร
=OFFSET(Ref, MATCH(\$B\$3, Id, 0)-1, 0, 1, 3)

วิเคราะห์สูตร

Target

=OFFSET(Ref, MATCH(\$B\$3, Id, 0)-1, 0, 1, 3)

ขอให้สังเกตเลข 0, 1, 3 ที่อยู่ด้านท้ายของสูตร Offset ก่อน

- เลข 0 กำหนดตำแหน่งของข้อมูล ให้เริ่มในแนว column เดียวกันกับ Ref
- เลข 1 คือ ความสูงของรายการข้อมูล ซึ่งต้องมีความสูงคงที่ 1 row เสมอ
- เลข 3 คือ ความกว้างของรายการ ซึ่งประกอบด้วย Id, Name, Amount จึงกำหนดให้กว้าง 3 column คงที่

ส่วนของสูตร MATCH(\$B\$3, Id, 0)-1 เป็นตัวช่วยกำหนดตำแหน่งรายการเดิมว่า Id ที่กรอกในเซลล์ B3 เป็นตำแหน่งลำดับที่เท่าใดใน Range ชื่อ Id สาเหตุที่ต้องลบผลที่ได้จากสูตร Match ออกเสีย 1 ตำแหน่ง ทั้งนี้เพื่อให้ตรงกับจำนวน row ที่ต้องนับ ถัดไปจากตำแหน่งของเซลล์ Ref

ดังนั้นสูตร Offset นี้ จึงทำหน้าที่ส่งรายการข้อมูลที่กรอกไว้ในตารางที่มีความสูง 1 row และกว้าง 3 column จาก Range Name ชื่อ Source ไปยัง Target ซึ่งมีขนาด ความสูงและความกว้างเดียวกัน เพียงแต่ Target จะขยับไปหาตำแหน่งรายการเดิมซึ่ง อยู่ถัดไปจาก Ref ตามที่นับได้ด้วยสูตร Match-1

ตัวอย่างนี้จะเกิด error ขึ้นทันที ถ้ารหัสที่กรอกลงไปในเซลล์ B3 ไม่ใช่รหัส Id ที่บันทึกเก็บไว้แล้วในตารางด้านขวา ดังนั้นถ้าต้องการป้องกันไม่ให้ชุดคำสั่ง SendData หยุดทำงานกลางคัน ให้แก้ชุดคำสั่งเล็กน้อยเป็นดังนี้

```
Sub SendData()
    On Error Resume Next
    MyVar = [Source]
    [Target] = MyVar
End Sub
```

On Error Resume Next ที่ใส่เพิ่มเป็นบรรทัดแรก จะทำหน้าที่ตรงกับความหมาย ซึ่ง แปลว่า "เมื่อเกิด error ขึ้น ให้กลับไปทำงานต่อ" ทำให้ชุดคำสั่ง SendData จะไม่เตือน error ให้เห็นอีกเลยแม้รหัสบรรทัดต่อไปจะไม่ทำงานก็ตาม

แทนที่จะใช้ On Error Resume Next ซึ่งไม่ได้ช่วยแก้ไขข้อมูลและไม่ได้เตือนให้ผู้ใช้ทราบว่าเกิดปัญหาแต่อย่างใด เราสามารถใช้ On Error Goto แทนดังนี้

```
Sub SendData()  
    On Error GoTo ResetInput  
    MyVar = [Source]  
    [Target] = MyVar  
    End  
ResetInput:  
    [Source] = "Invalid"  
End Sub
```

On Error Goto ทำหน้าที่ตรงกับค่าแปล คือ "เมื่อเกิด error ขึ้นให้ไปทำงานที่อื่นต่อ" ซึ่งในตัวอย่างนี้ กำหนดให้ข้ามไปทำงานต่อที่บรรทัด ResetInput โดยทำหน้าที่เปลี่ยนข้อมูลใน Source เป็นคำว่า Invalid เพื่อเตือนให้ทราบว่า ข้อมูลที่กรอกนั้นไม่ถูกต้อง

สังเกตว่าบรรทัด ResetInput: ต้องมีเครื่องหมาย colon : ต่อท้าย และนิยมเขียนโดยไม่จัดย่อหน้า

นอกจากนั้น ในกรณีที่ไม่มีเกิด error เราต้องป้องกันไม่ให้ชุดคำสั่งนี้ ทำงานในส่วนตั้งแต่บรรทัด ResetInput: จึงต้องใช้คำสั่ง End ไว้ก่อน เพื่อให้รหัสคำสั่งทำงานตามปกติ และจบไปเลย โดยไม่สนใจกับรหัสคำสั่งบรรทัดที่เหลือ

วิธีเลือกส่งข้อมูลตามลักษณะรายการเก่าใหม่

จาก 2 ตัวอย่างที่ผ่านไปแล้ว ตัวอย่างหนึ่งส่งให้รายการที่กรอก ไปบันทึกต่อท้าย ส่วนอีกตัวอย่างส่งให้รายการที่กรอก ไปบันทึกแก้ไขทับรายการเดิม จากนั้นจะหาทางนำทั้ง 2 ตัวอย่างรวมเข้าด้วยกัน ถ้าพบว่ารายการที่กรอกเป็นรหัสใหม่ ให้ส่งข้อมูลไปบันทึกต่อท้าย แต่ถ้าพบว่าเป็นรายการที่บันทึกไว้แล้วแสดงว่าเป็นรหัสเก่า ให้ส่งข้อมูลที่กรอกไปทับรายการเก่า ทั้งนี้ให้ใช้ชุดคำสั่งเดิม ไม่ต้องแก้ไขอะไรใน VBE

ชุดคำสั่ง : SendData

	A	B	C	D	E	F	G	H	I
1									
2		Id	Name	Amount		Target	Id	Name	Amount
3	Source	a002	boy	20		Ref	a001	a	10
4		↕	↕	↕			a002	b	20
5							a003	c	30
6		a004	d	40					
7									
8									
9									
10									
11									

สิ่งที่ต้องการ

ส่งข้อมูลที่กรอกลงในเซลล์ B3:D3 ไปเก็บในตารางด้านขวา โดยพิจารณาจากรหัส Id ที่กรอกในเซลล์ B3 ถ้าเป็นรหัส Id a002 ซึ่งเป็นรหัสเก่า ให้จัดส่งข้อมูลไปบันทึกทับรายการเดิมของรหัส a002 แต่ถ้าเป็นรหัสใหม่ที่ไม่เคยมีอยู่ในตารางด้านขวา เช่น รหัส Id a004 ให้จัดส่งข้อมูลไปบันทึกต่อท้ายรายการในตารางขวา

ขั้นตอนการสร้างงาน

1. สร้าง Range Name ชื่อ Source ให้กับเซลล์ B3:D3
2. สร้าง Range Name ชื่อ Ref ให้กับเซลล์ G3
3. สร้าง Range Name ชื่อ Id ให้กับเซลล์ G3:G10
4. สร้าง Formula Name ชื่อ Target ให้เป็นสูตร

```
=OFFSET( Ref,
  IF( COUNTIF( Id, $B$3 )>=1,
    MATCH( $B$3, Id, 0 )-1,
```


COUNTA(\$G:\$G)-1),
0, 1, 3)

วิเคราะห์สูตร

```
Target
=OFFSET( Ref,
  IF( COUNTIF( Id, $B$3 )>=1,
    MATCH( $B$3, Id, 0 )-1,
    COUNTA( $G:$G )-1),
  0, 1, 3 )
```

ขอให้สังเกตเลข 0, 1, 3 ที่อยู่ด้านท้ายของสูตร Offset ก่อน

- เลข 0 กำหนดตำแหน่งของข้อมูล ให้เริ่มในแนว column เดียวกันกับ Ref
- เลข 1 คือ ความสูงของรายการข้อมูล ซึ่งต้องมีความสูงคงที่ 1 row เสมอ
- เลข 3 คือ ความกว้างของรายการ ซึ่งประกอบด้วย Id, Name, Amount จึงกำหนดให้กว้าง 3 column คงที่

ส่วนของสูตร IF(COUNTIF(Id,\$B\$3)>=1, MATCH(\$B\$3,Id,0)-1,
COUNTA(\$G:\$G)-1)

- COUNTIF(Id,\$B\$3) ทำหน้าที่นับจำนวนรายการที่มี Id ที่กรอกใน B3
- ถ้า COUNTIF(Id,\$B\$3)>=1 เป็นจริง แสดงว่ามีรหัสรายการบันทึกอยู่แล้ว จึงให้ส่งข้อมูลไปบันทึกทับรายการเดิม ณ ตำแหน่งที่คำนวณจาก MATCH(\$B\$3,Id,0)-1
- ถ้า COUNTIF(Id,\$B\$3)>=1 เป็นเท็จ แสดงว่าเป็นรหัส Id ของรายการใหม่ จึงให้ส่งข้อมูลไปบันทึกต่อท้ายรายการเดิม ณ ตำแหน่งที่คำนวณจาก COUNTA(\$G:\$G)-1

โปรดสังเกตว่า สูตร IF ที่ใช้เป็นสูตรที่อยู่ใน Excel จึงช่วยให้ไม่ต้องเสียเวลากลับไปแก้ไขรหัส VBA เลยแม้แต่ครั้งเดียว

วิธีเรียกใช้รหัสเกี่ยวกับตาราง Excel

รหัส VBA ที่ใช้กันมากที่สุด เห็นจะเป็นรหัสที่เกี่ยวข้องกับการเรียกใช้พื้นที่ในตาราง Excel นี้แหละ ทั้งนี้เพื่อให้เราสามารถจัดการกับตัวแฟ้ม ชีท row column หรือ เซลล์ ได้ตามต้องการ และแทนที่จะพิมพ์รหัสซึ่งเกิดขึ้นจากการใช้ Macro Recorder ทางเดียว เราควรฝึกเขียนรหัส VBA ขึ้นมาใช้เองบ้าง เพราะแทนที่จะต้องใช้รหัสที่ได้จากการบันทึก เพื่อเลือกแฟ้ม แล้วเลือกชีท แล้วเลือกเซลล์ทีละขั้น เราสามารถใช้รหัสที่เขียนขึ้น พาสดไปยังเซลล์ที่ต้องการได้โดยตรง

ในรหัส VBA บรรทัดหนึ่งๆ จะไล่เรียงเนื้อหาจากซ้ายไปขวาโดยใช้เครื่องหมายจุดเป็นตัวแบ่ง ซึ่งเครื่องหมายจุดนี้ จะแปลว่า ของ ก็ได้ โดยจะไล่เรียงจากของใหญ่ไปของย่อย เช่น

```
Worksheets("Sheet1").Range("A1").Select
หรือ
Sheets("Sheet1").Range("A1").Select
```

รหัสบรรทัดนี้มีความหมายว่า "ให้เลือกเซลล์ A1 ของตารางที่อยู่ในชีทชื่อ Sheet1" และถ้าดูลำดับคำจากซ้ายไปขวา เวลาเขียนจะเรียงจากใหญ่ไปย่อย จากชีทไปยังเซลล์ โดยสิ่งต่างๆ ใน Excel ซึ่งเห็นได้บนจอและถูกเรียกชื่อได้ ถือว่าเป็น **Object** เช่น

- รหัสคำว่า Worksheets หมายถึง ตัวชีททั้งหมดหรือกลุ่มของชีท ซึ่งต้องลงท้ายด้วย s (เรียกว่าเป็น Collection ของชีท) โดยคำว่า Worksheets ยังหมายถึง ชีทที่เป็นตารางเท่านั้น (ถ้าอยากเรียกชีททั่วไป จะใช้คำว่า Sheets ซึ่งหมายรวมทั้ง Worksheet และ Chart sheet)
- คำว่า "Sheet1" เป็นชื่อชีท ที่เราต้องการเลือกออกมาจากชีททั้งหมดใน Worksheets ซึ่งในกลุ่มของชีทอาจมีหลายชีท ดังนั้น เราต้องระบุชื่อชีทที่ต้องการให้ถูกต้องชัดเจน

- Worksheets("Sheet1") จึงหมายถึง ชีทชื่อ Sheet1 ซึ่งเป็นชีทหนึ่งในชีททั้งหมดที่เป็นตาราง แต่ถ้าใช้ Sheets("Sheet1") จะหมายถึง ชีทชื่อ Sheet1 ซึ่งไม่จำเป็นต้องเป็นชีทที่เป็นตารางก็ได้

ส่วนคำว่า Select ท้ายสุด ถือว่าเป็น **Method** หมายถึงลักษณะอาการที่เราต้องการกระทำต่อตัว Object ดังนั้น Method Select จึงเท่ากับ สั่งให้คลิกเลือก

โดยทั่วไปในรหัสบรรทัดหนึ่งๆ รหัสเกี่ยวข้องกับ Object จะถูกเขียนไว้ท้ายสุด ต่อด้วยเครื่องหมายจุด แล้วต่อด้วย Method แต่ถ้าต้องการระบุถึงสภาพลักษณะของ Object ที่ต้องการ จะใช้ **Property** เขียนเรียงต่อจาก Object เช่น

```
Columns("B:B").ColumnWidth = 15.5
```

- Columns("B:B") เป็น Object ประเภท Column ซึ่งในที่นี้หมายถึงตัว Column B
- ColumnWidth เป็น Property หมายถึง ความกว้างของตัว Column ซึ่งในคำสั่งบรรทัดนี้ ทำให้ Column B มีความกว้าง 15.5 pixel

Property ต่างจาก Object ตรงที่ Property ไม่มีตัวตนให้มองเห็นหรือคลิกได้บนจอ แต่จะมีลักษณะเป็นสภาพ เช่น ขนาดความสูง ความกว้าง ความยาว ความหนา สี หรือสภาวะการป้องกัน

หมายเหตุ เรื่อง Object, Property, และ Method นี้เป็นเพียงชื่อเรียกศัพท์เทคนิคที่ยกขึ้นมา เพื่อแนะนำให้ทำความรู้จักกันไว้เท่านั้น ในขั้นนี้ยังไม่จำเป็นต้องสนใจเสียเวลาค้นหาว่า สิ่งใดบ้างเป็น Property ก็ได้ ขอเพียงระลึกไว้ว่า โครงสร้างรหัสคำสั่งแต่ละบรรทัด กำหนดความสำคัญจากซ้ายไปขวา เรียงจากสิ่งใหญ่ไปย่อยต่อไปเรื่อยๆ และถ้าอยากรู้ว่า VBA มีรหัสเป็นอย่างไร ขอให้ทดลองใช้ Recorder นั้นแหละ เมื่อคลิกเลือกลงไปในตาราง แล้วคลิกสั่งบนเมนู ก็จะเกิดบรรทัดคำสั่ง เรียง Object, Property, Method ขึ้นจากซ้ายไปขวาให้เอง

วิธีใช้รหัสจัดการกับพื้นที่ตาราง

ตัวอย่างรหัสต่อไปนี้แทบทั้งหมด เป็นตัวอย่างซึ่งได้จาก VBE Help โดยใช้คำว่า Name ในการค้นหา

- Worksheets(1).Visible = False
ซ่อนชีทแรก
- Worksheets("Sheet1").Protect password:="forall"
ป้องกันชีทชื่อ Sheet1 โดยใช้รหัสว่า forall
- Worksheets("Sheet1").Activate
เลือกชีทชื่อ Sheet1
- ActiveSheet.PrintOut
สั่งพิมพ์ชีทที่กำลังใช้งานอยู่
- Worksheets("Sheet1").Range("A1").Value = 100
ส่งเลข 100 ลงไปในเซลล์ A1 ของชีทชื่อ Sheet1
- Range("A1:H8").Formula = "=Rand()
สร้างสูตร =Rand() ลงไปในเซลล์ A1:H8 ของชีทที่กำลังใช้งานอยู่
- Worksheets(1).Range("Criteria").ClearContents
ลบข้อมูลในพื้นที่ที่มีชื่อ Range Name ว่า Criteria
- Worksheets(1).Cells(1, 1).Value = 24
ส่งเลข 24 ลงไปในชีทแรก ณ ตำแหน่งเซลล์ที่อยู่ row ที่ 1 คัดกับ

column ที่ 1 ของพื้นที่ในชีทแรก ซึ่งหมายถึงเซลล์ A1

- `ActiveSheet.Cells(2, 1).Formula = "=Sum(B1:B5)"`
สร้างสูตร `=Sum(B1:B5)` ลงไปในเซลล์ A2 ของชีทที่กำลังใช้งานอยู่
- `Worksheets(1).Range("C5:C10").Cells(1, 1).Formula = "=Rand()"`
สร้างสูตร `=Rand()` ลงไปในเซลล์ C5 ของชีทแรก (C5 คือ เซลล์ที่อยู่ในตำแหน่ง row ที่ 1 ตัดกับ column ที่ 1 ของพื้นที่ C5:C10)
- `Selection.Offset(3, 1).Range("A1").Select`
เลือกเซลล์ที่อยู่ถัดจากเซลล์ที่กำลังเลือกอยู่ ห่างลงไป 3 row และ
ถัดไปด้านขวา 1 column (รหัสแบบ Offset นี้จะเรียกเซลล์ที่ต้องการว่า
`Range("A1")` เสมอ) ดังนั้นถ้าเดิมอยู่ในเซลล์ A1 พอสั่งให้รหัสบรรทัดนี้
ทำงาน จะเลือกเซลล์ B4
- `Range("MyData")(4)=100`
ส่งเลข 100 ลงไปในเซลล์ที่ 4 ในพื้นที่ตารางที่มี Range Name ชื่อ
MyData โดยเซลล์ที่ 4 นี้จะนับจากซ้ายไปขวาแล้วนับต่อลงไปยัง row
ถัดลงไป แล้วนับต่อจากซ้ายไปขวาในพื้นที่ MyData
- `Range("MyData")(1,3)=100`
ส่งเลข 100 ลงไปในเซลล์ที่อยู่ ณ ตำแหน่ง row ที่ 1 ตัดกับ column ที่
3 ในพื้นที่ตารางที่มี Range Name ชื่อ MyData

VBA กับการตัดสินใจ

ถ้าเราต้องคอยควบคุมการทำงานของชุดคำสั่งทั้งหมด ตั้งแต่ต้นจนจบด้วยตนเอง คงเสียเวลาและเป็นเรื่องน่าเบื่อมิใช่น้อย ยิ่งมีขั้นตอนสลับซับซ้อนมากขึ้นเท่าใด ยิ่งต้องคอยให้ความใส่ใจ ใช่ว่าความระมัดระวังในการตัดสินใจเลือกสั่ง run macro ชุดคำสั่งที่ถูกต้อง เพราะเมื่อสั่งให้ macro ทำงานไปแล้ว ไม่มีทางที่จะสั่ง undo เหมือนกับที่ใช้คำสั่งบนเมนู ให้งานที่ทำไปแล้วคืนกลับสู่สภาพเดิมได้อีก

แทนที่จะต้องคอยตัดสินใจควบคุมการทำงานของ macro ในทุกขั้นตอนด้วยตนเอง เราสามารถใช้รหัส VBA ต่อไปนี้ช่วยตัดสินใจเลือกสั่งงานแทน

1. **If...Then...Else Statements** เหมาะสำหรับใช้ตัดสินใจเลือกขั้นตอนง่ายๆ หรือมีเงื่อนไขและใช้ตัวแปรซับซ้อนต่อเนื่องกันหลายขั้น
2. **Select Case Statements** เหมาะสำหรับใช้ตัดสินใจได้แทนแบบ If เพียงแต่แบบ Select Case มีโครงสร้างที่ง่ายต่อการนำตัวแปรตัวเดิมไปใช้เปรียบเทียบ

ข้อควรระวัง

ในชุดคำสั่งแบบ Sub Procedure โปรดเลือกใช้รหัส If...Then...Else หรือ Select Case ช่วยเฉพาะการตัดสินใจควบคุมการสั่งงานของชุดคำสั่ง โดยพยายามหลีกเลี่ยงอย่างน่าไปใช้ในการเขียนรหัสคำนวณ หรือกำหนดค่าคงที่ใดๆลงไปใน VBE ทั้งนี้เพื่อช่วยให้ไม่ต้องย้อนกลับมาแก้ไขในตัวรหัสอีกในภายหลัง

ถ้าจำเป็นต้องใช้ผลลัพธ์จากการคำนวณ ควรใช้ Excel คำนวณแทน แล้วใช้ Range Name ส่งผลลัพธ์ที่คำนวณได้กลับมาใช้ตัดสินใจต่อไปใน VBE

If...Then...Else Statements

- โครงสร้างแบบสั้นบรรทัดเดียว

If condition **Then** [statements] [**Else** elstatements]

- โครงสร้างแบบหลายบรรทัด

If condition **Then**

[statements]

[**ElseIf** condition-n **Then**

[elseifstatements] ...

[**Else**

[elstatements]]

End If

โครงสร้างที่เห็นนี้นามาจาก VBA Help โดยพิมพ์คำว่า If ลงไปในพื้นที่ที่ใช้พิมพ์รหัสตามปกติ แล้วกดปุ่ม F1 จะพบคำอธิบายวิธีใช้ If โดยขอให้สังเกตจากโครงสร้างว่า ส่วนใดที่เขียนไว้ภายในเครื่องหมายวงเล็บ [] ถือว่า เป็นส่วนที่จะเขียนหรือไม่ก็ได้ ส่วนที่เขียนด้วยตัวเข้มหนา เป็นส่วนที่ต้องเขียนเสมอ ดังนั้นโครงสร้างข้างต้น จึงมีส่วนที่จำเป็นต้องใช้ทั่วไปเพียงดังนี้

- กรณีใช้ตัดสินใจเฉพาะ **True**

If condition **Then** statements

หรือ

If condition **Then**

statements

End If

- กรณีใช้ตัดสินใจทั้ง **True** และ **False**

If condition **Then** statements **Else** elstatements

หรือ

If *condition* **Then**

statements

else

elsestatements

End If

Select Case Statement

Select Case *testexpression*

[**Case** *expressionlist-n*

[*statements-n*]] ...

[**Case Else**

[*elsestatements*]]

End Select

ตัวอย่างที่ 1

สมมติ ในแฟ้มมีเซลล์หนึ่งที่ตั้งชื่อว่า Source ไว้แล้ว โดยเราอยากให้ใช้ค่าใน Source เป็นตัวควบคุมการตัดสินใจเลือกให้ชุดคำสั่งที่ต้องการทำงานต่อไป

- ถ้า Source มีค่าเท่ากับ 100 พอดี ให้เลือกชุดคำสั่งชื่อ SendData1 ทำงาน
- ถ้า Source มีค่าไม่เท่ากับ 100 ให้เลือกชุดคำสั่งชื่อ SendData2 ทำงาน

```
If [Source] = 100 Then SendData1 Else SendData2
```

หรือ

```
If [Source] = 100 Then
```

```
    SendData1
```

```
Else
```

```
    SendData2
```



```

End If
หรือ
Select Case [Source]
    Case 100
        SendData1
    Case Else
        SendData2
End Select

```

ตัวอย่างที่ 2

ขอใช้ค่าจาก Source เช่นเดียวกับตัวอย่างแรก เพียงแต่แทนที่จะตรวจสอบว่า Source มีค่าเท่ากับ 100 หรือไม่เท่านั้น คราวนี้ให้เปรียบเทียบค่า Source ต่อไปอย่างละเอียด ดังนี้

- ถ้า Source มีค่ามากกว่า 100 ให้เลือกชุดคำสั่งชื่อ SendData1a ทำงาน
- ถ้า Source มีค่ามากกว่า 80 ให้เลือกชุดคำสั่งชื่อ SendData1b ทำงาน
- ถ้า Source มีค่ามากกว่า 60 ให้เลือกชุดคำสั่งชื่อ SendData1c ทำงาน
- ถ้า Source มีค่ามากกว่า 40 ให้เลือกชุดคำสั่งชื่อ SendData1d ทำงาน
- ถ้า Source มีค่ามากกว่า 20 ให้เลือกชุดคำสั่งชื่อ SendData1e ทำงาน
- ถ้า Source มีค่าอื่นๆ ให้เลือกชุดคำสั่งชื่อ SendData2 ทำงาน

ในกรณีที่มีเงื่อนไขซับซ้อนมากขึ้นนี้ ถ้าจะใช้ If...Then...Else Statements จะต้องเขียน ElseIf ซ้อนกันหลายชั้น แต่เนื่องจากทุกเงื่อนไขที่ใช้นั้น ใช้ค่าจาก Source เดียวกันตลอด จึงน่าจะเลือกใช้ Select Case Statement จะเขียนได้ง่าย มีโครงสร้างที่ชัดเจน และช่วยให้ย้อนกลับมาแก้ไขได้สะดวกกว่า

```

Select Case [Source]
    Case Is > 100
        SendData1a
    Case Is > 80
        SendData1b
    Case Is > 60

```

```

        SendData1c
    Case Is > 40
        SendData1d
    Case Is > 20
        SendData1e
    Case Else
        SendData2
End Select

```

IIf Function

ในกรณีที่ต้องการใช้สูตร IF ในโครงสร้างสูตรแบบเดียวกันกับที่ใช้สูตรใน Excel เราสามารถใช้สูตร IIF ใน VBA โดยมีโครงสร้างสูตร ดังนี้

IIf(expr, truepart, falsepart)

ตัวอย่าง

ถ้า Source มีค่ามากกว่า 1000 ให้ปรับค่าเหลือเท่ากับ 100

```
[Source] = IIf([Source] > 1000, 100, [Source])
```

โปรดสังเกตว่า เราใช้ IIF แบบสูตรเพื่อคำนวณหาผลลัพธ์ แต่ไม่ได้ใช้เพื่อสั่งให้รหัสคำสั่งอื่นทำงานโดยตรง ดังนั้นฟังก์ชันของ IIF จึงใช้เครื่องหมายเท่ากับเซลล์หรือตัวแปรที่ต้องการรับค่า

Choose Function

Choose(index, choice-1[, choice-2, ... [, choice-n]])

Choose ใช้คำนวณหาผลลัพธ์ได้เช่นเดียวกับ IIF แต่ Choose ใช้เลข index ซึ่งเป็นเลขจำนวนเต็ม ช่วยในการตัดสินใจหาคำตอบที่เขียนต่อไว้ในสูตร

ตัวอย่าง

ถ้า Source มีค่าเป็นเลข 1-5 ให้เปลี่ยนค่าใน Source เป็นตัวอักษร a, b, c, d, e ตามลำดับ

```
[Source] = Choose([Source], "a", "b", "c", "d", "e")
```

วิธีนำสูตรของ Excel มาใช้กับ VBA

แทนที่จะต้องเริ่มต้นเรียนรู้โครงสร้างสูตรของ VBA ขอแนะนำให้นำสูตร Excel ที่เราค้นเคยได้อยู่แล้วมาใช้แทนกันดีกว่า โดยพิมพ์คำว่า Application.WorksheetFunction ตามด้วยจุด นำหน้าสูตร Excel เช่น

```
[Source] = Application.WorksheetFunction.Round([Source], 2)
```

ปรับตัวเลขใน Source โดยปัดขึ้นเป็นเลขทศนิยม 2 หลัก

เดิม 12.345 จะปัดขึ้นเป็น 12.35

แต่ถ้าใช้สูตร Round ใน VBA

```
[Source] = Round([Source], 2)
```

เดิม 12.345 จะปัดเป็น 12.34

จะเห็นว่า สูตร Round ของ VBA คำนวณให้คำตอบต่างจาก Round ของ Excel โดยสูตร Round ของ VBA จะมีหลักพิจารณาเพิ่มเติมว่า ถ้าเลขหลักที่ต้องการเป็นเลขคู่อยู่แล้วจะไม่ปัดขึ้น

หากต้องการค้นหาว่ามีสูตร Excel ใดที่สามารถนำมาใช้ใน VBA ได้บ้าง ให้ค้นหาจาก VBA Help โดยค้นหาจากคำว่า List of Worksheet Functions Available to Visual Basic แล้วขอให้สังเกตว่าไม่มีสูตร If (โดยให้หันมาใช้ IIF ของ VBA แทน) หรือค้นหาจากพื้นที่ที่ใช้เขียนรหัส ให้พิมพ์คำว่า Application.WorksheetFunction แล้วพอพิมพ์เครื่องหมายจุดต่อท้ายค่า จะพบว่า VBE เปิดจอลิสต์ไล่ลำดับชื่อสูตรที่มีอยู่ เพื่อให้เลือกได้โดยตรงในขณะที่พิมพ์ให้ทันที

วิธีสร้างสัญญาณเตือน...ภัย

สัญญาณเตือน ถือเป็นสิ่งสำคัญอย่างยิ่งที่จะทำให้เราสบายใจมากขึ้น เพราะหากรหัส VBA ที่กำลังทำงานอยู่นั้น มีหลายขั้นตอน หรือกำลังจะทำขั้นตอนที่สำคัญห้าม ผิดพลาด ซึ่งก่อนที่จะเข้าสู่ขั้นตอนดังกล่าว ควรมีสัญญาณเตือนแสดงขึ้นบนจอ เพื่อ บอกให้เราทราบว่า ที่ผ่านไบนั้นเป็นขั้นตอนใดที่ทำเสร็จไปแล้ว ขณะนี้กำลังจะทำ ขั้นตอนใดต่อไป หรือเมื่อพบว่า จากขั้นตอนที่ทำไปแล้วนั้น ทำให้สถานการณ์ต่างไป จากเดิม เราจะยอมให้รหัสทำงานต่อไปอีกหรือไม่

MsgBox Function

MsgBox(prompt[, buttons] [, title])

ตัวอย่างการใช้งาน

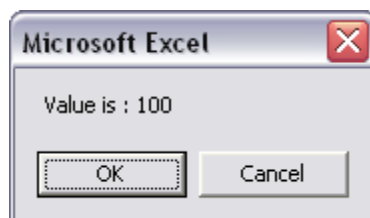
1. MsgBox "Value is : " & [Source]

แสดงค่าในเซลล์ชื่อ Source



2. MyVar = MsgBox("Value is : " & [Source], vbOKCancel)

If MyVar = vbOK Then [Target] = [Source]



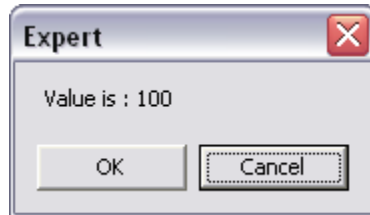
แสดงค่าในเซลล์ชื่อ Source แล้วถ้ากดปุ่ม OK

จะส่งค่าจาก Source ไปเก็บไว้ที่ Target

สังเกตว่า หลัง MsgBox มีวงเล็บเพื่อทำงานแบบสูตร

3. ถ้าแก้ไขเพิ่มเติมรหัสบรรทัดแรกในข้อ 2 เป็น

```
MyVar = MsgBox("Value is : " & [Source], vbOKCancel  
+ vbDefaultButton2, "Expert")
```



สังเกตว่า คราวนี้ปุ่ม Cancel เป็นปุ่มที่พร้อมใช้แทน และมีคำว่า Expert แทน คำว่า Microsoft Excel ในส่วนที่เป็น Title ด้านบน

เราสามารถเลือกใช้ปุ่มได้หลายประเภท โดยใช้รหัสต่อไปนี้แทนลงไปในส่วนของ buttons

- vbOKOnly
- vbOKCancel
- vbAbortRetryIgnore
- vbYesNoCancel
- vbYesNo
- vbRetryCancel
- vbCritical
- vbQuestion
- vbExclamation
- vbInformation

ในกรณีที่ใช้ปุ่มหลายปุ่ม ให้ใช้รหัสต่อไปนี้บอกต่อท้ายรหัสปุ่ม เพื่อเลือกให้เป็นปุ่มที่พร้อมใช้งานเมื่อกดปุ่ม Enter

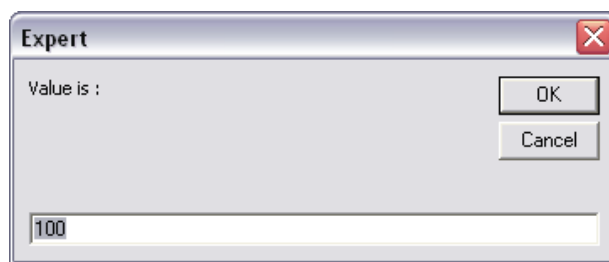
- vbDefaultButton1
- vbDefaultButton2
- vbDefaultButton3

InputBox Function

```
InputBox(prompt[, title] [, default])
```

ตัวอย่างการใช้งาน

```
[Source] = InputBox("Value is : ", "Expert", [Source])
```



InputBox แสดงค่าที่บันทึกไว้ในเซลล์ Source และเปิดให้บันทึกค่าใหม่แทนค่าเดิม

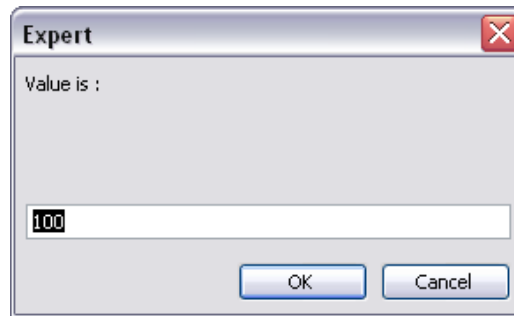
InputBox Method

นอกจากจะใช้ InputBox แบบที่เป็นสูตรแล้ว ยังมี InputBox แบบ Method อีกด้วย ซึ่งสามารถใช้เลือกบันทึกเป็นตัวเลข ตัวอักษร หรือสูตร โดยมีโครงสร้างดังนี้

```
InputBox(Prompt, Title, Default,,,, Type)
```

ตัวอย่างการใช้งาน

```
[Source] = Application.InputBox("Value is : ", "Expert", [Source], , , , 1)
```



- ต้องพิมพ์คำว่า Application. นำหน้า InputBox
- สังเกตว่า ตำแหน่งของปุ่ม OK Cancel จะวางไว้แนวนอน
- ค่าของ Type เพื่อควบคุมค่าที่ใช้ในการบันทึก

0	A formula
1	A number
2	Text (a string)
4	A logical value (True or False)
8	A cell reference, as a Range object
16	An error value, such as #N/A
64	An array of values
- ถ้าต้องการใช้ Type 2 อย่างพร้อมกัน ให้นำเลข Type มาบวกกัน เช่น ถ้าต้องการเปิดรับค่าที่เป็นตัวเลขหรือตัวอักษร ให้ใส่เลข 3 (=1+2)

รหัสอื่นๆที่เกี่ยวข้องกับสัญญาณเตือน

- Application.DisplayAlerts = False
Workbooks("BOOK1.XLS").Close
Application.DisplayAlerts = True
ยกเลิกการเตือนให้ save changes เมื่อปิดแฟ้ม
- Application.ScreenUpdating = False
....
Application.ScreenUpdating = True

ยกเลิกการปรับภาพที่แสดงการทำงานใดๆให้เห็นบนจอ

- Application.StatusBar = "Please be patient..."

Workbooks.Open filename:="LARGE.XLS"

Application.StatusBar = False

ในระหว่างที่กำลังเปิดแฟ้ม ให้แสดงข้อความให้เห็นบน Status Bar
ด้านซ้ายล่างของจอ และเมื่อเปิดแฟ้มเสร็จแล้วให้ลบข้อความออก

วิธีสั่งให้ VBA ทำงานทวนซ้ำหลายรอบ

คอมพิวเตอร์จะทำหน้าที่สมกับที่เป็นคอมพิวเตอร์ ต่อเมื่อเราสามารถสั่งให้คอมพิวเตอร์ทำงานเองตั้งแต่ต้นจนจบ และสามารถทำงานซ้ำแล้วซ้ำอีกได้ตามต้องการ โดยลักษณะการทำงานทวนซ้ำ (Looping) นี้มีหลายแบบ ได้แก่

1. การทวนซ้ำตามจำนวนรอบที่เรากำหนด
2. การทวนซ้ำตามจำนวนข้อมูล
3. การทวนซ้ำจนสมบูรณ์ตามเงื่อนไข

การทวนซ้ำตามจำนวนรอบที่เรากำหนด

For...Next Statements เป็นรหัสที่ใช้สั่งให้ทำงานทวนซ้ำตามจำนวนรอบที่เรากำหนด โดยมีโครงสร้างการเขียนดังนี้

```
For counter = start To end [Step step]
    [statements]
[Exit For]
    [statements]
Next [counter]
```

- counter เป็นตัวแปรที่กำหนดขึ้น ส่วนมากจะกำหนดชื่อตัวแปรเป็น i, j, k
- start เป็นเลขเริ่มต้นของรอบแรก
- end เป็นเลขสุดท้ายของรอบสุดท้าย
- step เป็นช่วงของเลขที่ต้องการให้บวกเพิ่มให้กับ counter ในรอบถัดไป ถ้าละส่วนของ step ไม่กำหนดลงไป จะถือว่า step=1
- statements เป็นรหัสคำสั่งที่ต้องการให้ทำงานซ้ำ
- Exit For ใช้กับกรณีที่ต้องการใช้เงื่อนไขให้เลิกทำงานวนซ้ำ

ตัวอย่าง

ต้องการสั่งพิมพ์สลิปเงินเดือนให้กับพนักงานทุกคน

	A	B	C	D	E
1					
2		Id	Name	Amount	
3		a001	a	1000	
4		a002	b	2000	
5		a003	c	3000	
6		a004	d	4000	
7		a005	e	5000	
8					
9		PaymentSlip			
10	Choice >	2	a002	b	2000
11					
12					
13	Total >	5			
14					
15		Choice	=Sheet1!\$B\$10		
16		MyData	=Sheet1!\$B\$3:\$D\$7		
17		Total	=Sheet1!\$B\$13		
18		C10 : '=INDEX(MyData,Choice,1)			
19					

```
For i = 1 To [Total]
```

```
    [Choice] = i
```

```
    ActiveWindow.SelectedSheets.PrintPreview
```

```
    ActiveWindow.SelectedSheets.PrintOut
```

```
Next i
```

- Total เป็น Range Name ในตารางที่เก็บรายละเอียดเงินเดือนพนักงาน โดยมีสูตร =CountA(ตารางชื่อพนักงานทั้งหมด) เพื่อหายอดจำนวนพนักงานทั้งหมด
- For i = 1 To [Total]
ทำหน้าที่สั่งให้เริ่มทำงานทวนซ้ำ ตั้งแต่รอบที่ 1 ถึง รอบสุดท้ายเท่าจำนวนพนักงานทั้งหมด
- Choice เป็น Range Name ใช้สำหรับรับเลขที่พนักงาน เพื่อใช้เลขที่พนักงานนี้ไปค้นหาข้อมูลของพนักงานเลขที่นั้นๆ นำมาแสดงในใบสลิปเงินเดือน

- ในตารางที่เตรียมไว้เป็นแบบสลิปเงินเดือน ให้ใช้สูตร Index เพื่อดึงข้อมูลของพนักงานตามเลขที่แสดงไว้ใน Choice เช่น ถ้าต้องการชื่อพนักงานที่เก็บไว้ใน column แรก ให้ค้นหาข้อมูลมาแสดงในสลิปโดยใช้สูตร

$$=Index(MyData,Choice,1)$$
- [Choice] = i
 ทำหน้าที่ส่งค่า i ไปที่เซลล์ Choice จึงทำให้สูตร Index คำนวณตามแล้วสูตร Index จะดึงข้อมูลของพนักงานตามเลขที่ของ i มาแสดงในใบสลิป ทั้งนี้ระบบการคำนวณต้องเป็น Automatic อยู่แล้วด้วย (แนะนำให้เพิ่มรหัสว่า Calculate ต่อท้ายบรรทัดนี้ เพื่อสั่งให้เพิ่มคำนวณ โดยไม่ต้องห่วงว่าเพิ่มเป็น Automatic หรือไม่)
- ActiveWindow.SelectedSheets.PrintPreview
 ActiveWindow.SelectedSheets.PrintOut
 ทำหน้าที่แสดงภาพสลิปบนหน้าจอให้เห็นก่อน รอให้เราคลิกสั่งปิด Print Preview จากนั้นจึงสั่งให้พิมพ์สลิปโดยอัตโนมัติ
 (ถ้าต้องการให้พิมพ์สลิปทันทีโดยไม่ต้องหยุดแสดงภาพที่จะพิมพ์บนจอ ให้ใช้รหัสคำสั่งบรรทัด PrintOut เพียงบรรทัดเดียว)
- Next i
 ทำหน้าที่สั่งให้วนกลับไปเพิ่ม $i = i + 1$ เป็นรายการถัดไป ขอแนะนำให้เขียนตัว i ต่อท้าย Next ไว้เสมอ เพื่อแสดงว่า Next ตัวนี้เป็นชุดของ i
- หากต้องการปรับปรุงรหัสให้ดีขึ้น แทนที่จะให้เริ่มจากเลข 1 ไปจนถึงคนสุดท้ายตามค่าที่ส่งมาจากตาราง ควรใช้ InputBox กำหนดค่า i และ Total เพื่อจะได้เลือกสั่งพิมพ์จากรายการใดก็ได้ และเมื่อพิมพ์เสร็จแล้วควรย้อนค่าใน Choice กลับเป็นเลขเริ่มต้นเดิม หรือเป็นเลข 1 เพื่อแสดงรายการแรก ดีกว่าปล่อยให้ตารางแสดงข้อมูลรายการสุดท้ายค้างไว้

การทวนซ้ำตามจำนวนข้อมูล

For Each *element* **In** *group*

[*statements*]

[Exit For]

[*statements*]

Next [*element*]

- *element* เป็นตัวแปรที่เรากำหนดให้แทนแต่ละส่วนของ *group* โดยจะตั้งชื่อตัวแปรนี้เป็นค่าว่าอะไรก็ได้
- *group* เป็นกลุ่มของ Object เช่น พื้นที่ตารางที่มีหลายเซลล์ หรือแฟ้มที่มีหลายชีท

ตัวอย่าง

ต้องการปรับค่าในตารางที่ตั้งชื่อว่า Source ถ้าเซลล์ใดในตารางนี้ มีค่าน้อยกว่า 0 ให้ปรับค่าให้เป็น 0

```
For Each c In [Source]
    c.Select
    If c.Value < 0 Then c.Value = 0
Next c
```

c เป็นตัวแปรที่เราตั้งขึ้นเพื่อใช้แทน แต่ละเซลล์ในตารางที่มีชื่อว่า Source

การทวนซ้ำจนสมบูรณ์ตามเงื่อนไข

โครงสร้างรหัสซึ่งสามารถนำมาใช้ทำงานทวนซ้ำจนเงื่อนไขสมบูรณ์มีหลายแบบ เช่น

Do [{**While** | **Until**} *condition*]

[*statements*]

[Exit Do]

[*statements*]

Loop

Do
 [statements]
 [Exit Do]
 [statements]
Loop [{While | Until} condition]

ตัวอย่าง

ต้องการปรับเพิ่มค่าของเลขที่บันทึกไว้ใน column โดยให้ทำงานเฉพาะพื้นที่เซลล์ที่มีค่าบันทึกไว้ เป็นตารางติดต่อกันไปเริ่มจากเซลล์ A1

```
Range("A1").Select
Do Until Selection.Value = ""
    Selection.Value = Selection.Value + 1
    Selection.Offset(1, 0).Select
Loop
```

- Do Until Selection.Value = ""
 สั่งให้ทวนซ้ำไปจนกว่าจะพบว่าเซลล์เป็นช่องว่าง
- Selection.Value = Selection.Value + 1
 ปรับค่าในเซลล์ที่เลือกให้มีค่าเพิ่มอีก 1
- Selection.Offset(1, 0).Select
 เลือกเซลล์ row ถัดไปข้างล่าง ใน column เดิม
- ถ้าใช้ Do While Selection.Value <> ""
 สั่งให้ทวนซ้ำไปเรื่อยๆ ตราบใดที่เซลล์ที่เลือกไม่เป็นเซลล์ว่าง

นอกจากนี้ยังมีรหัสแบบ While...Wend แต่ไม่แนะนำให้ใช้ เพราะไม่ยืดหยุ่นเท่า Do...Loop

วิธี Exit

หากต้องการออกจากวงจรการทำงานซ้ำให้ใช้รหัส If ช่วยตามแบบต่อไปนี้

- If Selection.Value > 10 Then Exit For
- If Selection.Value > 10 Then Exit Do
- If Selection.Value > 10 Then Exit Sub

หมายเหตุ Exit Sub เป็นการสั่งให้ออกจากชุดคำสั่งที่กำลังทำงานอยู่ แล้วกลับไปทำงานตามคำสั่งถัดไปในชุดคำสั่งที่เป็นตัวสั่งเรียกชุดคำสั่งอื่นทำงาน (ถ้าใช้ End จะหยุดการทำงานทั้งหมด)

เมนูคำสั่งใน Excel ซึ่งทำงานทวนซ้ำโดยไม่ต้องใช้ VBA

ถ้างานใดสามารถใช้เมนูคำสั่งของ Excel ได้โดยตรง ควรใช้คำสั่งของ Excel ที่มีอยู่แล้วดีกว่า เพราะนอกจากจะสะดวกกว่าการใช้ VBA แล้ว ยังทำงานรวดเร็วกว่ามาก

- **F5 > Special**
ใช้ในการค้นหาเซลล์ลักษณะต่างๆในตาราง
- **Home > Find and Select**
ใช้ค้นหาค่าและเปลี่ยนแปลงค่าในเซลล์
- **Data > What-If Analysis > Goal Seek**
ใช้คำนวณย้อนกลับ เพื่อหาค่าตัวแปรใหม่ จนได้ผลลัพธ์ตามต้องการ
- **Data > What-If Analysis > Data Table**
ใช้ส่งค่าตัวแปรไปคำนวณซ้ำ
- **ปุ่ม F4**
สั่งให้ Excel ทำงานตามคำสั่งล่าสุดซ้ำอีกครั้ง

วิธีสร้างสูตรเพื่อนำมาใช้ในงานเฉพาะด้าน

ลองคิดถึงข้อเสียของการสร้างสูตรยาวๆ ลงไปในตาราง ยิ่งตารางสูตรมีขนาดใหญ่ มากขึ้นเท่าใด จะยิ่งทำให้แฟ้มมีขนาดใหญ่ขึ้นมากเท่านั้น แล้วถ้าต่อมามต้องสร้างชีทที่มีตารางสูตรซ้ำกันอีกหลายๆ ชีท ไม่ใช่แค่แฟ้มจะใหญ่ขึ้นเพียงอย่างเดียว แต่ยังทำให้ จำเป็นต้องคอยติดตามแก้ไขสูตร ที่พิมพ์ไว้ในทุกชีทให้เหมือนกันไปด้วย

ถ้าต้องการย่อสูตรยาวๆ ให้สั้นลง โดยหาทางใช้ VBA สร้างสูตรขึ้นมาใช้ จะมีแนวทางการใช้สูตรที่สร้างด้วย VBA (Function VBA หรือ User Defined Function - UDF) 2 แบบ คือ

1. สร้างสูตรเพื่อใช้กับแฟ้มใดแฟ้มหนึ่งโดยเฉพาะ
2. สร้างสูตรเพื่อใช้กับแฟ้มใดก็ได้ (Add-in)

หมายเหตุ แทนที่จะคิดใช้ Function VBA ขอให้พยายามใช้สูตรสำเร็จรูปของ Excel หรือนำสูตรสำเร็จรูปมาใช้คำนวณร่วมกัน เพื่อคำนวณหาคำตอบที่ต้องการให้ได้ก่อน เพราะสูตรสำเร็จรูปจะคำนวณเร็วกว่ามาก

Function VBA มิได้สร้างใน Sub Procedure แต่ให้สร้างไว้ใน Function Procedure ซึ่งมีโครงสร้างของชุดคำสั่งที่เป็นสูตร ดังนี้

```
Function ชื่อสูตร(ชื่อตัวแปร1,ชื่อตัวแปร2,,,ชื่อตัวแปรn)
รหัสที่ใช้ในการคำนวณ
End Function
```

วิธีสร้างสูตรเพื่อใช้กับแฟ้มใดแฟ้มหนึ่งโดยเฉพาะ

1. ให้เปิด VBE ขึ้นมาแล้ว **Insert > Module**
2. พิมพ์คำว่า function ตามด้วยชื่อสูตร และวงเล็บของชื่อตัวแปร
3. กดปุ่ม Enter จะพบคำว่า End Function พิมพ์เป็นบรรทัดสุดท้ายให้เอง
4. ให้พิมพ์รหัสที่ต้องการใช้คำนวณลงไปในบรรทัดระหว่าง Function...End Function

ตัวอย่างที่ 1 : สูตรที่จะแสดงสูตรของเซลล์ที่ต้องการ

```
Function FML(cell)
    FML = cell.Formula
End Function
```

เมื่อต้องการใช้สูตรในตาราง เช่น ต้องการแสดงสูตรจากเซลล์ A1 ให้พิมพ์ =FML(A1)

ตัวอย่างที่ 2 : สูตรที่ใช้ตรวจสอบว่ามีสูตรในเซลล์นั้นๆหรือไม่

```
Function HasFML(cell)
    HasFML = cell.HasFormula
End Function
```

เมื่อต้องการใช้สูตรในตาราง เช่น ต้องการตรวจสอบว่าเซลล์ A1 มีสูตรสร้างไว้หรือไม่ ให้พิมพ์ =HasFML(A1) จะได้คำตอบเป็น True/False

ตัวอย่างที่ 3 : สูตรหาค่ารวมเฉพาะเซลล์ที่เป็นตัว Bold

```
Function BoldSum(rngCells)
    BoldSum = 0
    For Each c In rngCells
        If c.Font.Bold Then BoldSum = BoldSum + c.Value
    Next c
End Function
```

เมื่อต้องการหาค่ารวมของตัวเลขในตาราง A1:B10 โดยให้รวมเลขจากเซลล์ที่ใช้ตัวหนาเท่านั้น ให้สร้างสูตร =BoldSum(A1:B10)*Now()/Now()

สาเหตุที่ต้องนำ Now()/Now() คูณเข้าไป เพื่อให้สูตรนี้คำนวณใหม่ทุกครั้งเมื่อเรากดปุ่ม F9

วิธีสร้างสูตรเพื่อใช้กับแฟ้มใดก็ได้ (Add-in)

Add-in มิได้มีรหัสอื่นใดที่แตกต่างจากตัวอย่างข้างต้น เพียงแค่สั่ง save แฟ้มโดยเลือก Save as type เป็น Microsoft Office Excel Add-In (*.xla) จะได้แฟ้มใหม่ในชื่อเดียวกับแฟ้มเดิม โดยแฟ้มใหม่ที่เกิดขึ้นนี้มีนามสกุล xla

ก่อนที่จะ save เป็น xla ควรปรับปรุงแฟ้มในส่วนต่อไปนี้

- เนื่องจากแฟ้ม xla ที่จะนำมาใช้งานต่อไปนั้น ใช้เป็นแฟ้มที่นำสูตร Function VBA ที่เราสร้างขึ้นมาใช้งาน โดยไม่ได้ใช้เนื้อที่ชีทแสดงขึ้นบนจอ ดังนั้นเพื่อให้ประหยัดหน่วยความจำ ควรลบชีททิ้งให้หมดจนเหลือเพียงชีทเดียว เพื่อให้แฟ้ม xla มีขนาดเล็กที่สุดเท่าที่จะทำได้
- ควรทำคำอธิบายประกอบสูตรแต่ละสูตร โดยสั่ง **Developer > Macros** ซึ่งจะไม่พบชื่อ Macro ที่เป็น Function VBA แสดงไว้ ให้พิมพ์ชื่อสูตรตามที่ตั้งชื่อไว้ลงไปในช่วง Macro name จะพบว่าปุ่ม Options แสดงขึ้น ให้คลิกเข้าไปแล้วพิมพ์คำอธิบายสูตรลงในช่อง Description (คำอธิบายสูตรนี้จะแสดงประกอบตัวสูตร เมื่อใช้คำสั่ง Formulas > Insert Function จะพบสูตรแสดงไว้ในส่วนของ User Defined)
- ควรใช้ตารางในชีทเดียวที่เหลืออยู่ สร้างตัวอย่างวิธีการใช้สูตร และบันทึกชื่อผู้สร้างไว้ด้วย แล้วสั่ง **Protect Sheet**
- ควรใช้คำสั่ง **File > Info > Properties** บันทึกรายละเอียดโดยย่อสั้นๆ ของแฟ้มสูตรลงในช่อง Title และบันทึกรายละเอียดเพิ่มเติม โดยเฉพาะข้อมูลเรื่องลิขสิทธิ์ ลงในช่อง Comments (เมื่อนำ Add-in มาใช้งาน จะพบข้อความที่บันทึกใน Properties นี้แสดงให้เห็นในช่วง Add-Ins Available แทนที่จะแสดงแค่ชื่อแฟ้มให้เห็นเท่านั้น)
- ควรสั่ง **Protect Workbook** เพื่อป้องกันไม่ให้แก้ไข Properties ที่บันทึกรายละเอียดไว้
- ในส่วนของรหัสที่สร้างไว้ใน VBE ควรสั่ง **Tools > VBAProject Properties > Protection** แล้วกาช่อง **Lock project for**

viewing (เพื่อกันไม่ให้เห็นโครงสร้างภายในแฟ้มว่ามีชื่ออย่างไร) และ
ใส่รหัสป้องกันไว้ด้วย

วิธีติดตั้งและข้อควรระวังในการใช้ Add-in

1. copy แฟ้ม Add-in ที่ต้องการไปเก็บไว้ใน folder ใดก็ได้
2. เปิด Excel แล้วสั่ง Developer > Add-ins > Browse หาแฟ้ม xla ที่เก็บไว้ จะพบว่ามีชื่อ Add-in ที่ต้องการถูกกาเครื่องหมายถูกไว้
3. เปิดแฟ้มที่มีสูตร Add-in สร้างไว้ จะพบว่าสูตรทำงานตามต้องการ (แต่ถ้าเปิดแฟ้มไว้ก่อนที่จะ Browse ในข้อ 2 จะพบว่าสูตร error ต้องเข้าไปกด F2 ที่เซลล์สูตรแล้วกด Enter เพื่อกระตุ้นให้สูตรทำงาน)

ถ้าในแฟ้ม ไม่ได้ใช้สูตร Add-in ควรเลิกใช้ Add-in เพื่อทำให้ไม่เปลืองหน่วยความจำ โดยสั่ง **Developer > Add-ins > ตัดกาเครื่องหมายถูกทิ้ง** ต่อเมื่อต้องการใช้ Add-in จึงกลับมาถูกด้านหน้าเฉพาะ Add-in ที่ต้องการใช้งาน

ถ้าจำโครงสร้างสูตรไม่ได้ ให้ดูได้จาก **Insert Function** แล้วเลือกสูตรแบบ User defined พอคลิกชื่อสูตรจะพบคำอธิบายแสดงไว้ด้านล่างสุด

รหัส VBA ที่น่าสนใจจาก www.Excel-VBA.com

Calculation

Application.Calculation = xlManual

ปรับระบบการคำนวณเป็น Manual เพื่อหยุดการคำนวณชั่วคราว

Application.Calculation = xlAutomatic

ปรับระบบการคำนวณเป็น Automatic

ActiveSheet.Calculate

สั่งคำนวณเฉพาะชีทที่กำลังใช้งานอยู่

CutCopyMode

Application.CutCopyMode=False

ล้างหน่วยความจำใน clipboard เพื่อลดภาระของเครื่อง

GoTo

Application.Goto Reference:=Range("V300")

หรือ

Range("V300").Select

หรือ

Application.Goto Reference:=Range("V300"), Scroll=True

เพื่อให้เซลล์ที่ถูกเลือกเป็นเซลล์แรกซ้ายบนสุดของจอ

Quit : ปิดโปรแกรม Excel

Application.Quit

ScreenUpdating

Application.ScreenUpdating = False

Application.ScreenUpdating = True

ควบคุมไม่ให้อัปเดตที่อยู่ระหว่างคำสั่ง 2 บรรทัดนี้ แสดงการเปลี่ยนแปลงใดๆ ให้เห็นบนหน้าจอ

Workbook

Workbooks.Open "suchAndSuch.xls"

เปิดแฟ้มชื่อ suchAndSuch.xls

Workbooks.Open Sheets("sheet1").Range("A1").Value

เปิดแฟ้มที่มีชื่อตามที่บันทึกชื่อไว้ในเซลล์ A1 ของ Sheet1

ThisWorkbook.Close

ปิดแฟ้มที่เก็บรหัสที่ทำงานนี้

ThisWorkbook.Saved = True

ThisWorkbook.Close

ปิดแฟ้มที่เก็บรหัสที่ทำงานนี้โดยไม่ต้อง save

ActiveWorkbook.Close

ปิดแฟ้มที่กำลังใช้งานนี้

ActiveWorkbook.Saved = True

ActiveWorkbook.Close

ปิดแฟ้มที่กำลังใช้งานนี้โดยไม่ต้อง save

Workbooks("Book1.xls").Close

ปิดแฟ้มชื่อ Book1.xls

Workbooks(Range("A1").Value).Close

ปิดแฟ้มชื่อตามที่ระบุไว้ในเซลล์ A1

ActiveWorkbook.Save

save แฟ้มที่กำลังใช้งาน

Workbooks("Book1.xls").Save

save แฟ้มชื่อ Book1.xls

Workbooks(Range("A1").Value).Save

save แฟ้มที่มีชื่อตามที่ระบุไว้ในเซลล์ A1

ActiveWorkbook.SaveAs "C:/suchAndSuch.xls"

save แฟ้มที่กำลังใช้งานตามชื่อและสถานที่เก็บตามต้องการ

Workbooks("Book1.xls").SaveAs "C:/suchAndSuch.xls"

save แฟ้มชื่อ Book1.xls ตามชื่อและสถานที่เก็บตามต้องการ

Kill "C:\myFile.xls"

ลบแฟ้มชื่อ myfile.xls ที่เก็บอยู่ใน Drive C

Worksheet

Sheets("Sheet1").Visible= xlVeryHidden

hide ชีทแรกแบบพิเศษ เพื่อให้หาจากเมนู Format > Sheet ไม่พบ

Sheets(Array("Sheet1", "Sheet2")).Select

เลือกชีทหลายชีทตามชื่อที่กำหนดพร้อมกัน

ActiveWindow.SelectedSheets.Visible = False

ซ่อนชีทที่เลือกไว้ทั้งหมดพร้อมกัน

Sheets("Balance").Delete

ลบชีทชื่อ Balance

Sheets.Add

insert ชีท 1 ชีท

Sheets.Add before:=Sheets("Balance")

insert ชีท 1 ชีท ไว้ก่อนชีทชื่อ Balance

Sheets.Add after:=Sheets(1)

insert ชีท 1 ชีท ไว้หลังชีทแรก

Sheets.Add After:=Sheets(Sheets.Count)

insert ชีท 1 ชีท ไว้หลังชีทสุดท้าย

Cell และ Range

Cells.Select

เลือกทุกเซลล์ในตาราง

Selection.CurrentRegion.Select

เลือกพื้นที่ตารางที่ติดต่อกันกับเซลล์ที่เลือกอยู่

Activecell.Row

Activecell.Column

คืนค่าเป็นเลขที่ของ row/column ของเซลล์ที่ใช้งานอยู่

Selection.Rows.Count

Selection.Columns.Count

คืนค่าเป็นจำนวน row/column ของตารางที่เลือก

Selection.CurrentRegion.Rows.Count

คืนค่าเป็นจำนวน row ของพื้นที่ตารางที่ติดต่อกันกับเซลล์ที่เลือกอยู่

Range("A1:A8").Formula = "=C8+C9"

สร้างสูตร =C8+C9, =C9+C10 และต่อไป ลงไปในเซลล์ A1:A8

Range("A1:A8").Formula = "=\$C\$8+\$C\$9"

สร้างสูตร =\$C\$8+\$C\$9 ลงไปในทุกเซลล์ของ A1:A8

Range("A8:G8").Select

เลือกพื้นที่ตาราง A8:G8 โดย A8 เป็นเซลล์เดี่ยวที่ active

รหัส VBA ที่น่าสนใจจาก www.MindSpring.com

<http://www.mindspring.com/~tflynn/excelvba.html>

Selecting

```
Sub SelectDown()
    Range(ActiveCell, ActiveCell.End(xlDown)).Select
End Sub

Sub Select_from_ActiveCell_to_Last_Cell_in_Column()
    Dim topCel As Range
    Dim bottomCel As Range
    On Error GoTo errorHandler
    Set topCel = ActiveCell
    Set bottomCel = Cells((65536), topCel.Column).End(xlUp)
    If bottomCel.Row >= topCel.Row Then
        Range(topCel, bottomCel).Select
    End If
    Exit Sub
errorHandler:
    MsgBox "Error no. " & Err & " - " & Error
End Sub

Sub SelectUp()
    Range(ActiveCell, ActiveCell.End(xlUp)).Select
End Sub

Sub SelectToRight()
    Range(ActiveCell, ActiveCell.End(xlToRight)).Select
End Sub

Sub SelectToLeft()
    Range(ActiveCell, ActiveCell.End(xlToLeft)).Select
End Sub
```



```

Sub SelectCurrentRegion()
    ActiveCell.CurrentRegion.Select
End Sub

Sub SelectActiveArea()
    Range(Range("A1"), ActiveCell.SpecialCells(xlLastCell)).Select
End Sub

Sub SelectActiveColumn()
    If IsEmpty(ActiveCell) Then Exit Sub
    On Error Resume Next
    If IsEmpty(ActiveCell.Offset(-1, 0)) Then Set TopCell = _
        ActiveCell Else Set TopCell = ActiveCell.End(xlUp)
    If IsEmpty(ActiveCell.Offset(1, 0)) Then Set BottomCell = _
        ActiveCell Else Set BottomCell = ActiveCell.End(xlDown)
    Range(TopCell, BottomCell).Select
End Sub

Sub SelectActiveRow()
    If IsEmpty(ActiveCell) Then Exit Sub
    On Error Resume Next
    If IsEmpty(ActiveCell.Offset(0, -1)) Then Set LeftCell = _
        ActiveCell Else Set LeftCell = ActiveCell.End(xlToLeft)
    If IsEmpty(ActiveCell.Offset(0, 1)) Then Set RightCell = _
        ActiveCell Else Set RightCell = ActiveCell.End(xlToRight)
    Range(LeftCell, RightCell).Select
End Sub

Sub SelectEntireColumn()
    Selection.EntireColumn.Select
End Sub

Sub SelectEntireRow()
    Selection.EntireRow.Select

```

```
End Sub
```

```
Sub SelectEntireSheet()
```

```
    Cells.Select
```

```
End Sub
```

```
Sub ActivateNextBlankDown()
```

```
    ActiveCell.Offset(1, 0).Select
```

```
    Do While Not IsEmpty(ActiveCell)
```

```
        ActiveCell.Offset(1, 0).Select
```

```
    Loop
```

```
End Sub
```

```
Sub ActivateNextBlankToRight()
```

```
    ActiveCell.Offset(0, 1).Select
```

```
    Do While Not IsEmpty(ActiveCell)
```

```
        ActiveCell.Offset(0, 1).Select
```

```
    Loop
```

```
End Sub
```

```
Sub SelectFirstToLastInRow()
```

```
    Set LeftCell = Cells(ActiveCell.Row, 1)
```

```
    Set RightCell = Cells(ActiveCell.Row, 256)
```

```
    If IsEmpty(LeftCell) Then _
```

```
        Set LeftCell = LeftCell.End(xlToRight)
```

```
    If IsEmpty(RightCell) Then _
```

```
        Set RightCell = RightCell.End(xlToLeft)
```

```
    If LeftCell.Column = 256 And RightCell.Column = 1 Then _
```

```
        ActiveCell.Select Else Range(LeftCell, RightCell).Select
```

```
End Sub
```

```
Sub SelectFirstToLastInColumn()
```

```
    Set TopCell = Cells(1, ActiveCell.Column)
```

```
    Set BottomCell = Cells(16384, ActiveCell.Column)
```

```
    If IsEmpty(TopCell) Then Set TopCell = TopCell.End(xlDown)
```

```

If IsEmpty(BottomCell) Then _
    Set BottomCell = BottomCell.End(xlUp)
If TopCell.Row = 16384 And BottomCell.Row = 1 Then _
    ActiveCell.Select Else Range(TopCell, BottomCell).Select
End Sub

Sub SelCurRegCopy()
    Selection.CurrentRegion.Select
    Selection.Copy
    Range("A17").Select ' Substitute your range here
    ActiveSheet.Paste
    Application.CutCopyMode = False
End Sub

```

Check Values

```

Sub ResetValuesToZero2()
    For Each n In Worksheets("Sheet1").Range("WorkArea1")
        If n.Value <> 0 Then
            n.Value = 0
        End If
    Next n
End Sub

Sub ResetTest1()
    For Each n In Range("B1:G13")
        If n.Value <> 0 Then
            n.Value = 0
        End If
    Next n
End Sub

Sub ResetTest2()
    For Each n In Range("A16:G28")
        If IsNumeric(n) Then

```

```

        n.Value = 0
    End If
Next n
End Sub

Sub ResetTest3()
    For Each amount In Range("I1:I13")
        If amount.Value <> 0 Then
            amount.Value = 0
        End If
    Next amount
End Sub

Sub ResetTest4()
    For Each n In ActiveSheet.UsedRange
        If n.Value <> 0 Then
            n.Value = 0
        End If
    Next n
End Sub

Sub ResetValues()
    On Error GoTo ErrorHandler
    For Each n In ActiveSheet.UsedRange
        If n.Value <> 0 Then
            n.Value = 0
        End If
TypeMismatch:
    Next n
ErrorHandler:
    If Err = 13 Then      'Type Mismatch
        Resume TypeMismatch
    End If
End Sub

```

```

Sub ResetValues2()
    For i = 1 To Worksheets.Count
        On Error GoTo ErrorHandler
        For Each n In Worksheets(i).UsedRange
            If IsNumeric(n) Then
                If n.Value <> 0 Then
                    n.Value = 0
ProtectedCell:
                End If
            End If
        Next n
ErrorHandler:
        If Err = 1005 Then
            Resume ProtectedCell
        End If
    Next i
End Sub

```

On Entry

```

Sub Auto_Open()
    ActiveSheet.OnEntry = "Action"
End Sub

Sub Action()
    If IsNumeric(ActiveCell) Then
        ActiveCell.Font.Bold = ActiveCell.Value >= 500
    End If
End Sub

Sub Auto_Close()
    ActiveSheet.OnEntry = ""
End Sub

```

Looping

'You might want to step through this using the "Watch" feature

```
Sub Accumulate()
Dim n As Integer
Dim t As Integer
    For n = 1 To 10
        t = t + n
    Next n
    MsgBox "    The total is " & t
End Sub
```

'This sub checks values in a range 10 rows by 5 columns

'moving left to right, top to bottom-----

```
Sub CheckValues1()
Dim rwIndex As Integer
Dim colIndex As Integer
    For rwIndex = 1 To 10
        For colIndex = 1 To 5
            If Cells(rwIndex, colIndex).Value <> 0 Then _
                Cells(rwIndex, colIndex).Value = 0
        Next colIndex
    Next rwIndex
End Sub
```

'Same as above using the "With" statement instead of "If"

```
Sub CheckValues2()
Dim rwIndex As Integer
Dim colIndex As Integer
    For rwIndex = 1 To 10
        For colIndex = 1 To 5
            With Cells(rwIndex, colIndex)
                If Not (.Value = 0) Then _
                    Cells(rwIndex, colIndex).Value = 0
            End With
        Next colIndex
    Next rwIndex
End Sub
```

```

        Next colIndex
    Next rwIndex
End Sub

'Same as CheckValues1 except moving top to bottom, left to right
Sub CheckValues3()
Dim colIndex As Integer
Dim rwIndex As Integer
    For colIndex = 1 To 5
        For rwIndex = 1 To 10
            If Cells(rwIndex, colIndex).Value <> 0 Then _
                Cells(rwIndex, colIndex).Value = 0
        Next rwIndex
    Next colIndex
End Sub

'Enters a value in 10 cells in a column and then sums the values
Sub EnterInfo()
Dim i As Integer
Dim cel As Range
Set cel = ActiveCell
    For i = 1 To 10
        cel(i).Value = 100
    Next i
cel(i).Value = "=SUM(R[-10]C:R[-1]C)"
End Sub

' Loop through all worksheets in workbook and reset values
' in a specific range on each sheet.
Sub Reset_Values_All_WSheets()
Dim wSht As Worksheet
Dim myRng As Range
Dim allwShts As Sheets
Dim cel As Range
Set allwShts = Worksheets

```

```

For Each wSht In allwShts
Set myRng = wSht.Range("A1:A5, B6:B10, C1:C5, D4:D10")
    For Each cel In myRng
        If Not cel.HasFormula And cel.Value <> 0 Then
            cel.Value = 0
        End If
    Next cel
Next wSht
End Sub

```

Test Values

```

' Tests the value in each cell of a column and if it is greater
' than a given number, places it in another column. This is just
' an example so the source range, target range and test value may
' be adjusted to fit different requirements.
Sub Test_Values()
Dim topCel As Range, bottomCel As Range, _
    sourceRange As Range, targetRange As Range
Dim x As Integer, i As Integer, numofRows As Integer
Set topCel = Range("A2")
Set bottomCel = Range("A65536").End(xlUp)
If topCel.Row > bottomCel.Row Then End
' test if source range is empty
Set sourceRange = Range(topCel, bottomCel)
Set targetRange = Range("D2")
numofRows = sourceRange.Rows.Count
x = 1
For i = 1 To numofRows
    If Application.IsNumber(sourceRange(i)) Then
        If sourceRange(i) > 1300000 Then
            targetRange(x) = sourceRange(i)
            x = x + 1
        End If
    End If
End For
End Sub

```


Next

End Sub

วิธีปรับ Excel ให้พร้อมต่อการใช้ VBA

การจัดเตรียมทั่วไป

1. ปรับให้ใช้งานแฟ้มที่มีรหัส VBA ได้ โดยสั่ง File > Excel Options > Trust Center > Trust Center Settings > Macro Settings > กาช่อง Disable all macro with notifications (เป็น Default ของ Excel 2007/2010)
2. ตอนเปิดแฟ้มจะมีคำถามขึ้นมาบนหน้าจอ ให้เลือก Enable Macros (ถ้าไม่ไวใจแฟ้มนั้น ให้เลือก Disable Macros)
3. Excel 2007/2010 สามารถกำหนด Trust Locations เพื่อระบุชื่อ Folder ที่เก็บแฟ้มที่ต้องการให้ Enable Macro เองทันทีเมื่อเปิดแฟ้ม
4. Excel 2010 สามารถกำหนด Trust Document เพื่อสั่งให้ Excel จดจำว่าแฟ้มที่เคยเปิดนั้นถูกสั่งให้ใช้ Macro อย่างไร
5. อย่าลืม Save แฟ้มที่มีรหัส VBA เป็นนามสกุล xlsm

การเตรียมพร้อมสำหรับใช้ VBA

Excel 2007 สั่ง Office Button > Excel Options > Popular > กาช่อง Show Developer tab in the Ribbon

Excel 2010 สั่ง File > Options > Customize Ribbon > กา Developer ที่แสดงในรายชื่อในจอด้านขวา

การปรับระบบแสดงผลภาษาไทยใน Visual Basic Editor (VBE)

ให้กดปุ่ม ALT+F11 เพื่อเปิด VBE ขึ้นมา แล้วสั่ง Tools > Options > Editor Format แล้วเลือก Font ภาษาไทย

Copyright

ถ้าผู้ใดประสงค์จะนำข้อมูลในเว็บ XLSiam.com นี้ ไปเผยแพร่โดยการพิมพ์แจก หรือทำสำเนาผ่านสื่อใดๆ เพื่อใช้ในการศึกษาของตนเอง หรือเพื่อนำไปแจกผู้อื่นเป็นวิทยาทานโดยไม่คิดมูลค่า และเป็นการกระทำที่ใช้การลงทุนลงแรงของตนเอง ผู้นั้นกรุณาพิมพ์ได้หรือทำสำเนาได้โดยไม่ต้องขออนุญาตแต่อย่างใด ขอเพียงระบุที่มาของข้อมูลเหล่านั้นไว้เสมอ

สำหรับข้อมูลส่วนของผู้ซึ่งร่วมใช้เว็บนี้ในการเขียนบทความ ถือเป็นสิทธิของผู้เขียนบทความ หรือผู้ให้คำตอบ ที่จะนำข้อมูลส่วนของตนเองไปใช้ตามที่ตนต้องการ และผู้นั้นสามารถเลือกที่จะกำหนดเงื่อนไขในลิขสิทธิ์ในส่วนข้อมูลของตนเอง ในการยินยอมให้ผู้อื่นนำข้อมูลของตนไปใช้ ให้แตกต่างไปจากข้อกำหนดในลิขสิทธิ์ของเว็บนี้หรือไม่ก็ได้ หากมิได้กำหนดเงื่อนไขใดๆ ไว้ ถือว่าให้เป็นไปตามข้อกำหนดของลิขสิทธิ์ของเว็บนี้ที่กำหนดไว้

ห้ามผู้อื่นใดนำข้อมูลบนเว็บ XLSiam.com นี้ไปเผยแพร่เพื่อประโยชน์ใดๆก็ตาม ที่เกี่ยวข้องกับ การมุ่งค้าหากำไรทั้งทางตรงและทางอ้อม หรือมีส่วนได้เสียหรือหวังผลอื่น ซึ่งมีไขความพึงพอใจจากการให้เพื่อเป็นวิทยาทาน

หากทำเพื่อการจำหน่ายหรือเกี่ยวข้องกับการแสวงหาผลประโยชน์อื่นใด ไม่ว่าจะเป็นการหารายได้หรือช่วยลดค่าใช้จ่ายของตนหรือบริษัทของตนที่ต้องรับภาระ เช่น นำเนื้อหาทั้งหมดหรือบางส่วนไปรวบรวมเป็นเล่มเพื่อขายร่วมกับเรื่องอื่น หรือนำไปใช้เป็นเอกสารหรือใช้ประกอบสื่อใดๆในการอบรมแทนที่จะลงทุนจัดทำตำราเอง หรือใช้การลงทุนลงแรงของผู้อื่น ซึ่งไม่ใช่การลงทุนลงแรงของตนเอง ขอสงวนลิขสิทธิ์ข้อมูลทั้งหมดในเว็บนี้ตามกฎหมาย

สมเกียรติ ฟุ่งเกียรติ

20 เมษายน 2556