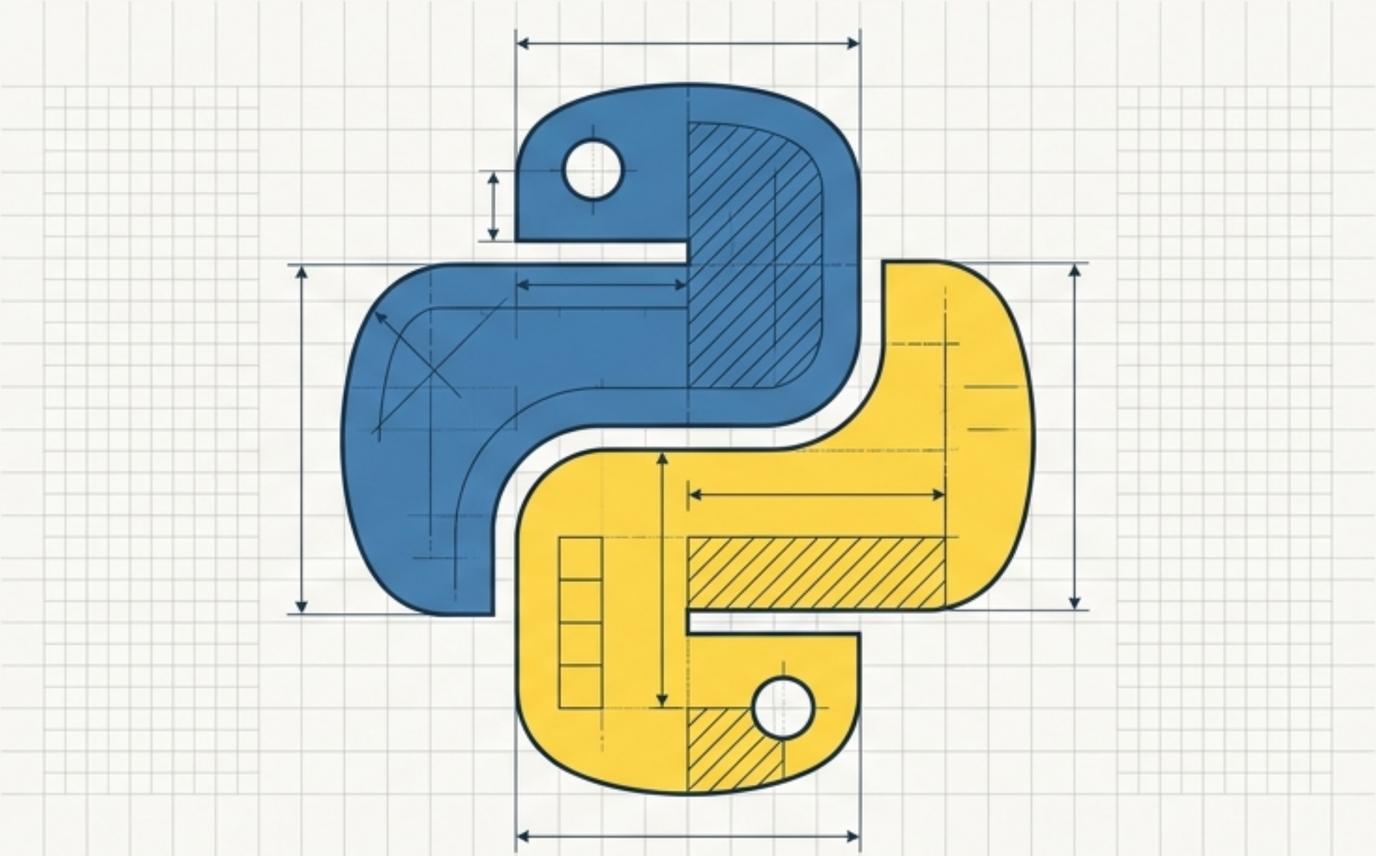


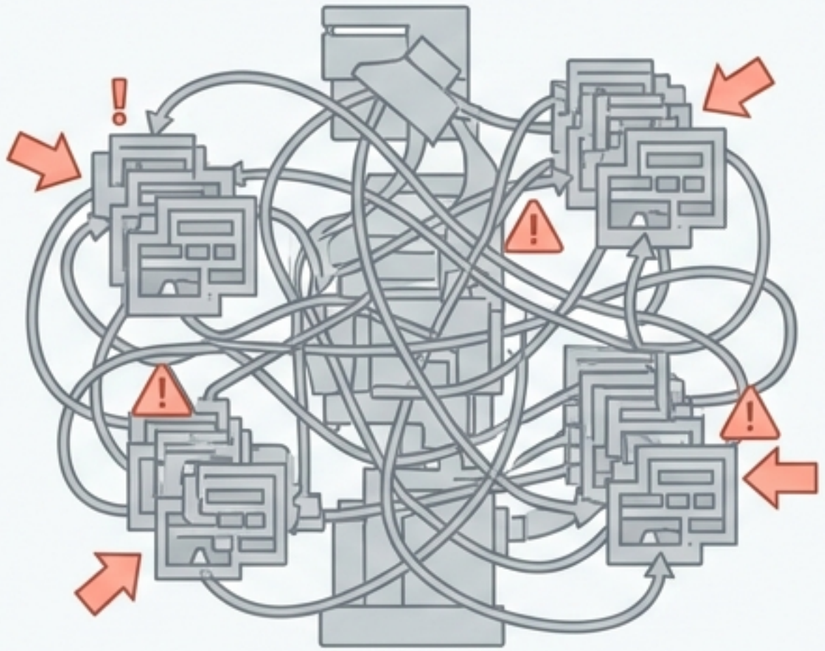
ฟังก์ชัน (Functions) ในภาษา Python

The Complete Visual Guide & Data Flow Blueprint

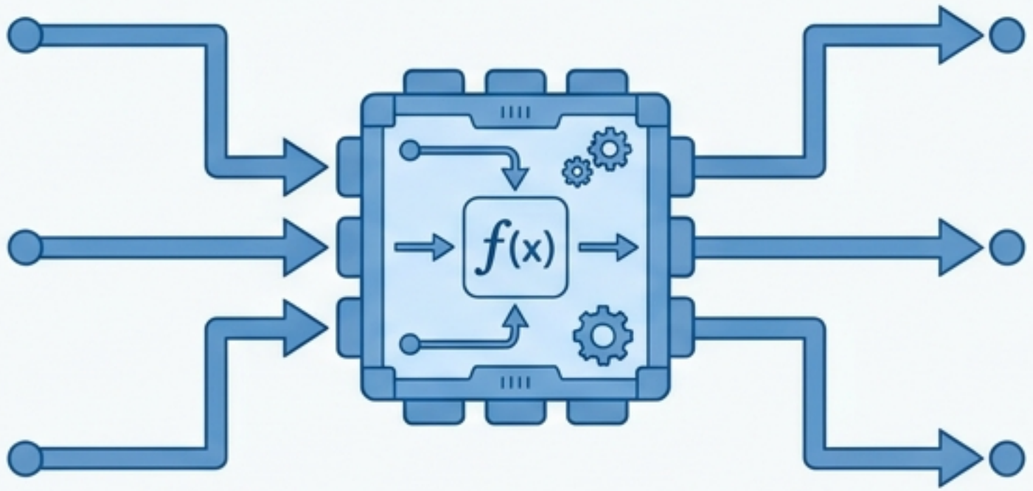
คู่มือฉบับภาพสำหรับการออกแบบและการจัดการ Flow ของข้อมูล

ปัญหาการเขียนโค้ดซ้ำซาก และกฎ DRY

ไม่มีฟังก์ชัน

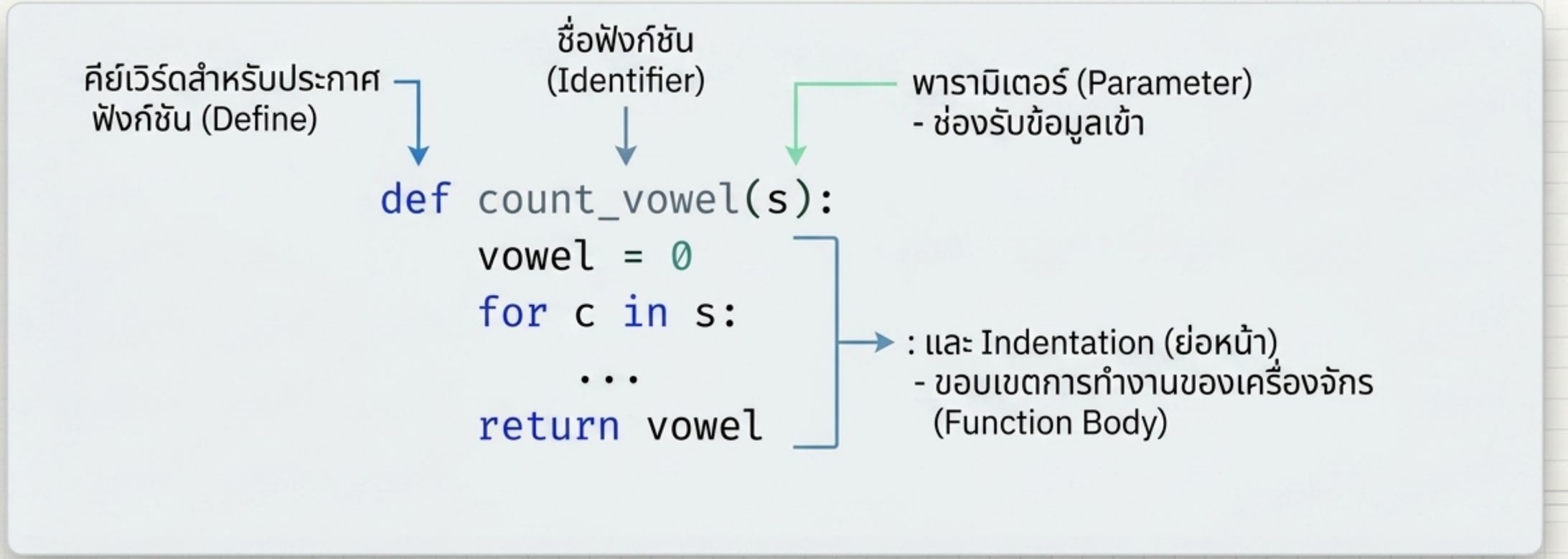


ใช้ฟังก์ชัน



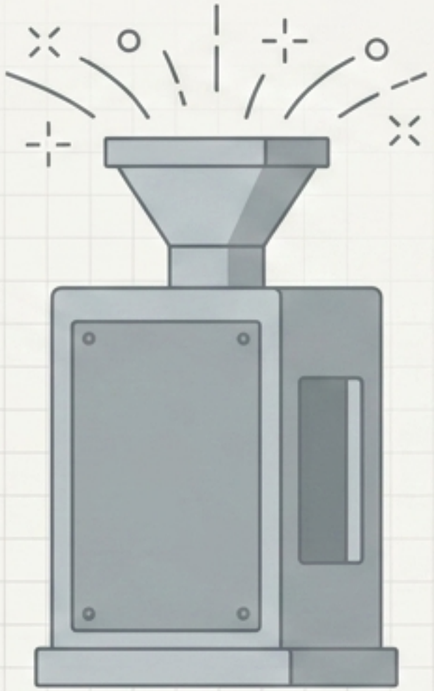
DRY (Don't Repeat Yourself) - แยกโค้ดที่มีการทำงานเหมือนๆ กันเป็นศูนย์กลาง แล้วเรียกใช้งานซ้ำ (Code Reuse) เพื่อความง่ายในการจัดการ

กายวิภาคของการสร้างฟังก์ชัน (Function Anatomy)



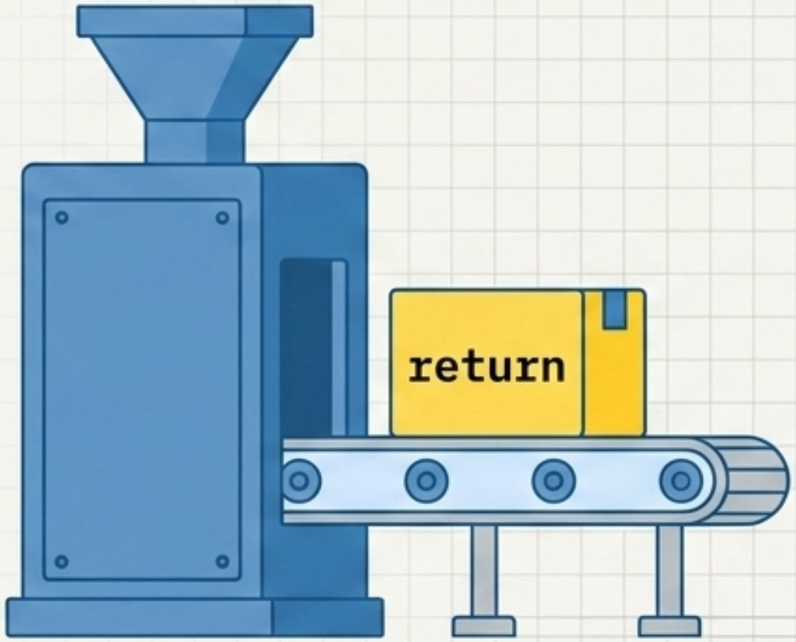
โพรซีเยอร์ (Procedure) vs ฟังก์ชันที่ส่งค่ากลับ (Return)

Machine A



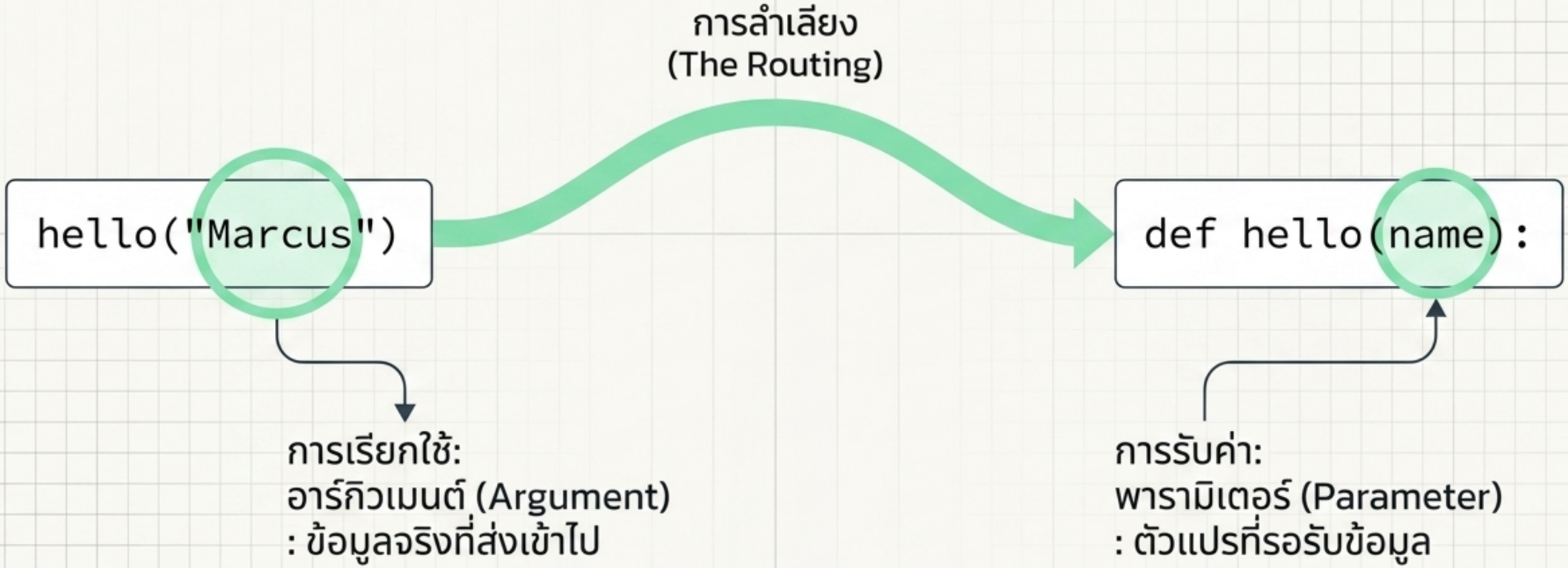
แบบไม่มี Return: ทำงานเสร็จสิ้นในตัว
เช่น hello(name) ที่แค่พิมพ์ข้อความออกหน้าจอ

Machine B



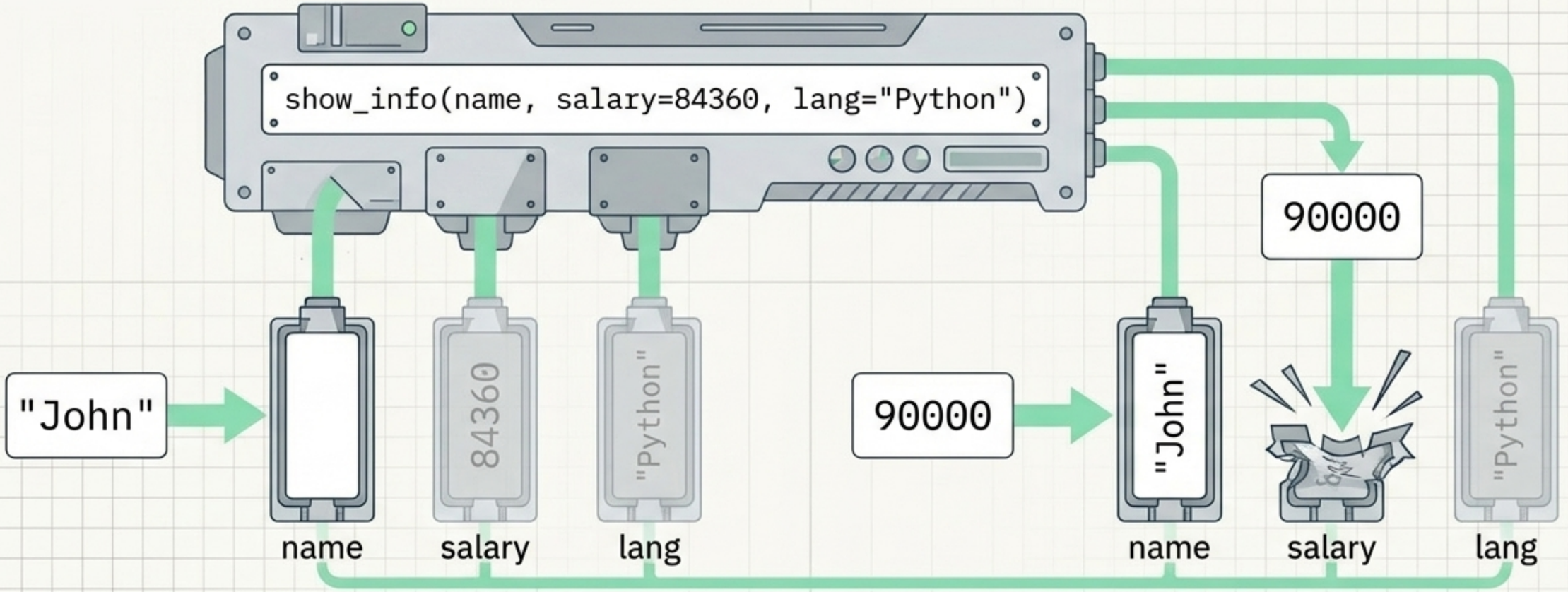
แบบมี Return: ส่งมอบผลลัพธ์กลับไปให้โปรแกรมนำไปใช้งานต่อ
เช่น area(w, h) ส่งคืนค่าพื้นที่ด้วยคำสั่ง return

วงจรการเรียกใช้งาน (The Function Call Flow)



วิวัฒนาการที่ 1: Default Argument

การกำหนดค่าเริ่มต้นให้ฟังก์ชันทำงานได้
แม้ผู้ใช้งานส่งข้อมูลมาไม่ครบ



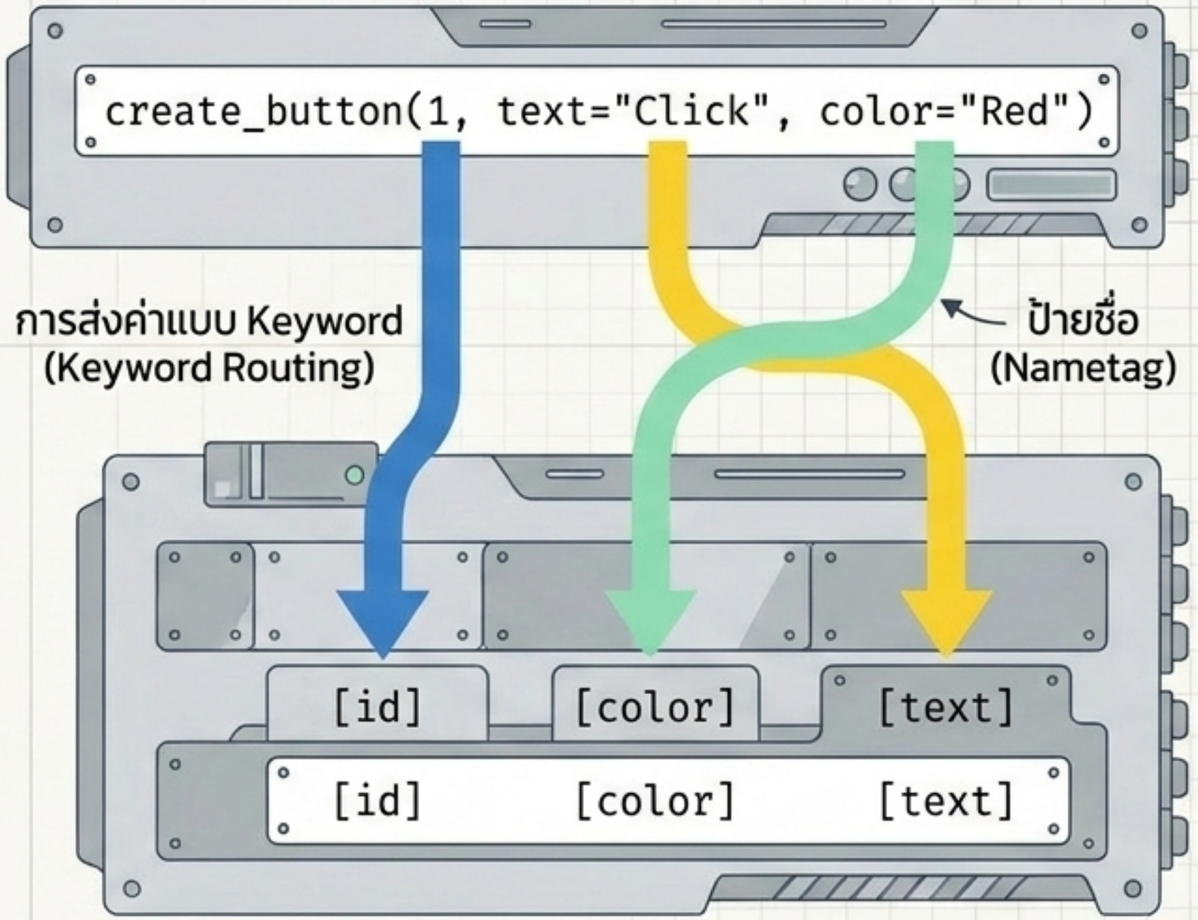
กฎเหล็ก: พารามิเตอร์ที่มีค่า Default ต้องอยู่ 'ด้านขวาสุด' หรือหลังพารามิเตอร์ปกติเสมอ!

วิวัฒนาการที่ 2: Keyword Arguments

ระบุชื่อพารามิเตอร์และส่งค่าแบบ argument = value

- ข้อดี: ไม่ต้องจำลำดับ! (Order Independent) ข้อมูลจะวิ่งไปไปเสียบถูกช่องอัตโนมัติตาม “ป้ายชื่อ” ที่แปะไว้
- ตัวอย่าง Built-in:

  `" print("A", "B", sep="-") "`



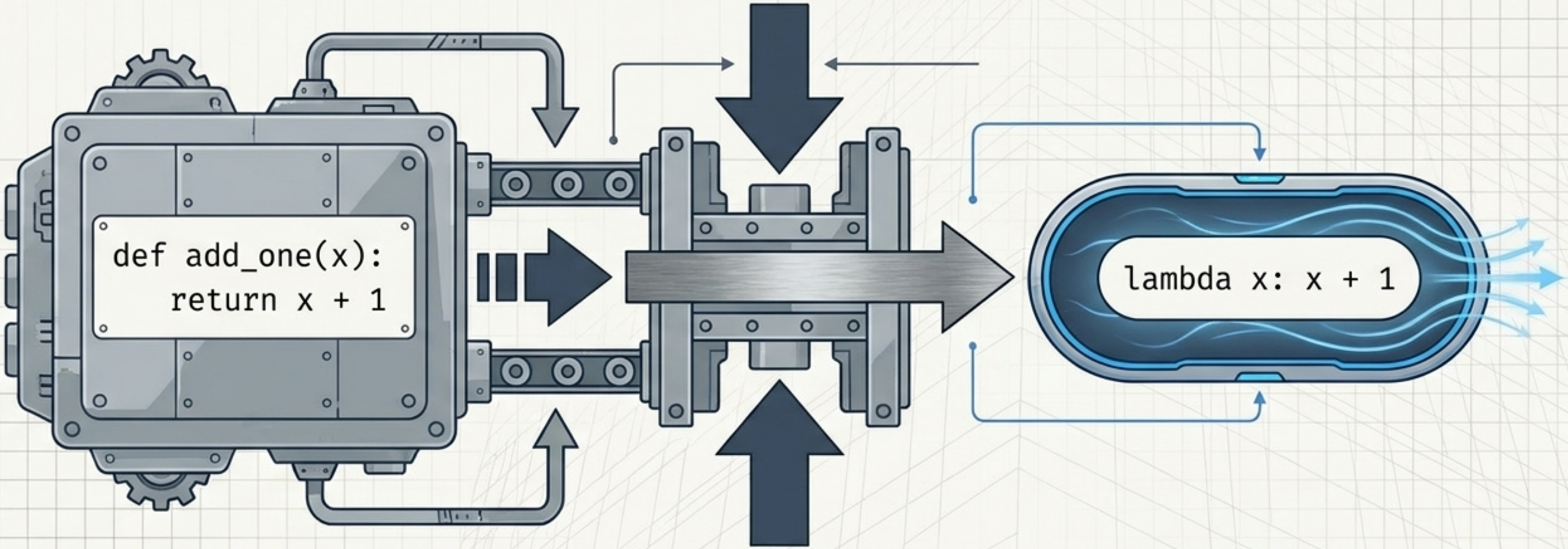
The Argument Flexibility Matrix

ประเภท (Type)	รูปแบบ (Syntax)	ต้องเรียงลำดับ? (Order Dependency)	บังคับใส่ค่า? (Required?)
Positional	`func(value)`	บังคับลำดับ ✓	บังคับใส่ ✓
Default	`func(param=value)`	บังคับลำดับ ✓ (ถ้าไม่ระบุชื่อ)	ไม่บังคับ ✗ (ใช้ค่าเริ่มต้น)
Keyword	`func(param=value)`	ไม่บังคับลำดับ ✗	ไม่บังคับ ✗ (ถ้ามี default)

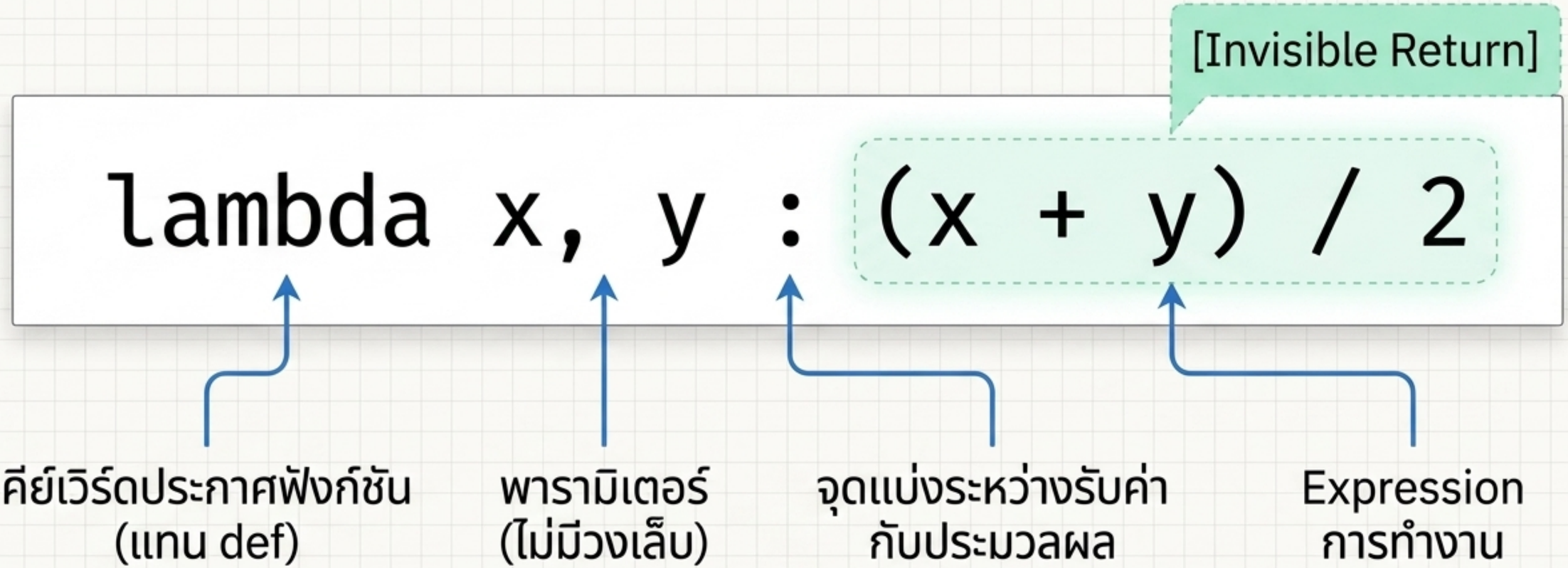
Note: Keyword arguments นิยมใช้เมื่อฟังก์ชันมีพารามิเตอร์จำนวนมากเพื่อลดความสับสน

The Paradigm Shift: เข้าสู่โลกของ Lambda Expressions

Anonymous Function (ฟังก์ชันไม่มีชื่อ) ทำงานขนาดเล็ก ใช้โค้ดเพียงบรรทัดเดียว (Single Expression) และรีเทิร์นค่าอัตโนมัติ

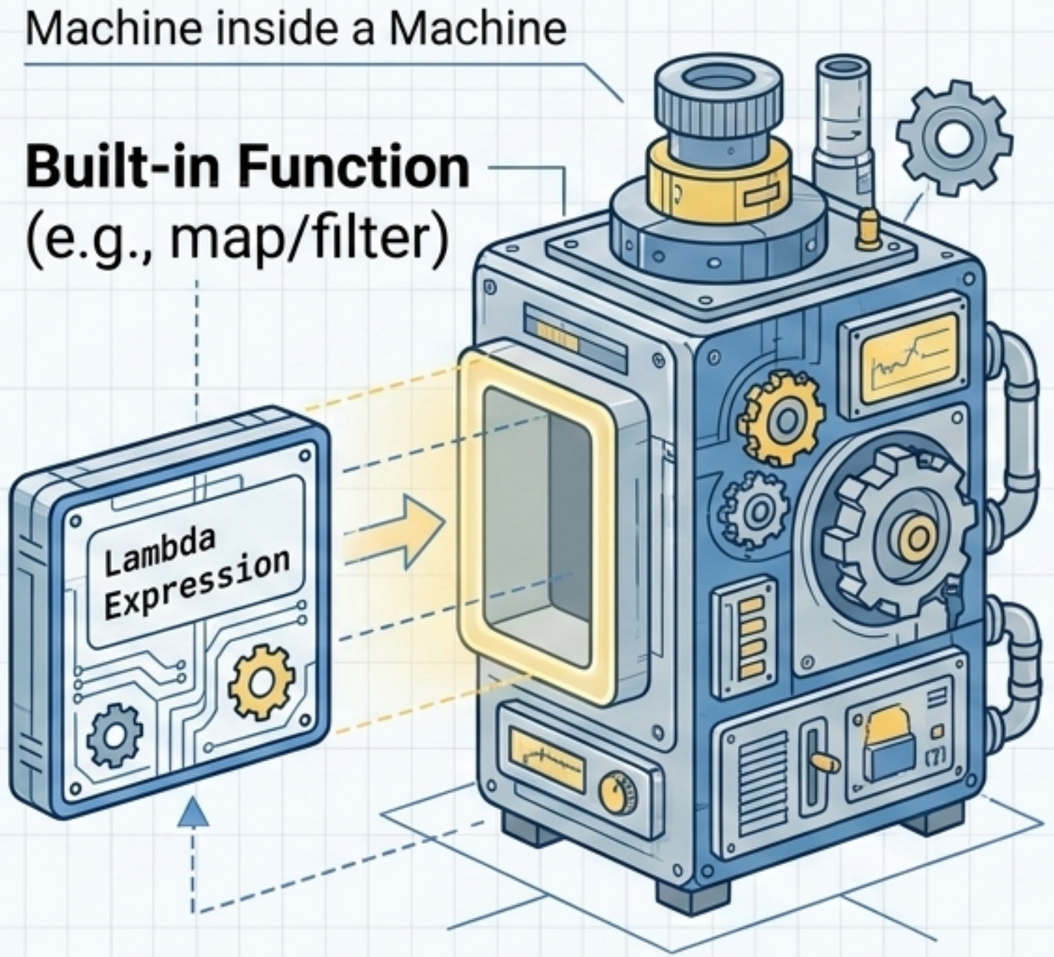


กายวิภาคของ Lambda (Lambda Anatomy)



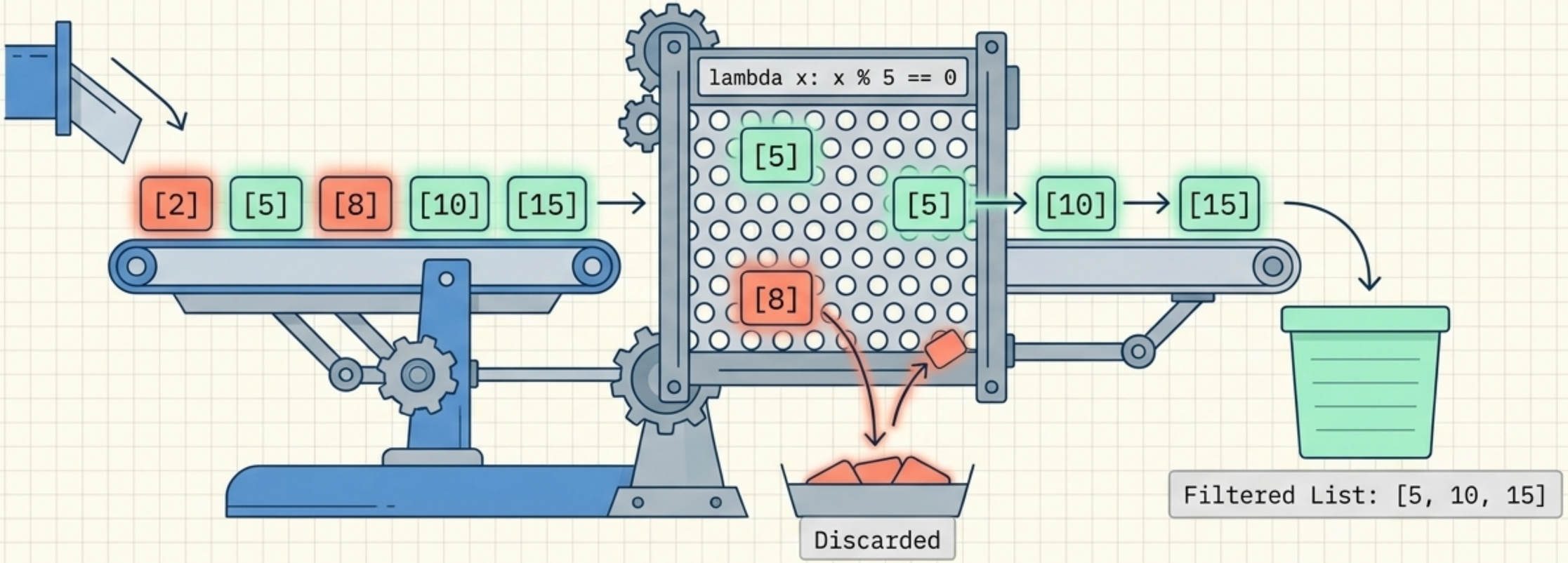
ฟังก์ชันระดับสูง (Higher-Order Functions)

- ใน Python, ฟังก์ชันถือเป็นข้อมูลชนิดหนึ่ง (Data)
- เราสามารถ "ส่งฟังก์ชันเข้าไปในฟังก์ชันอื่น" ในฐานะอาร์กิวเมนต์ได้!
- Lambda คือ "ตลับลอจิก" (Logic Cartridge) ขนาดเล็กที่เหมาะที่สุดสำหรับเสียบเข้า built-in functions



Visualizing **filter()**: ตัวกรองข้อมูล

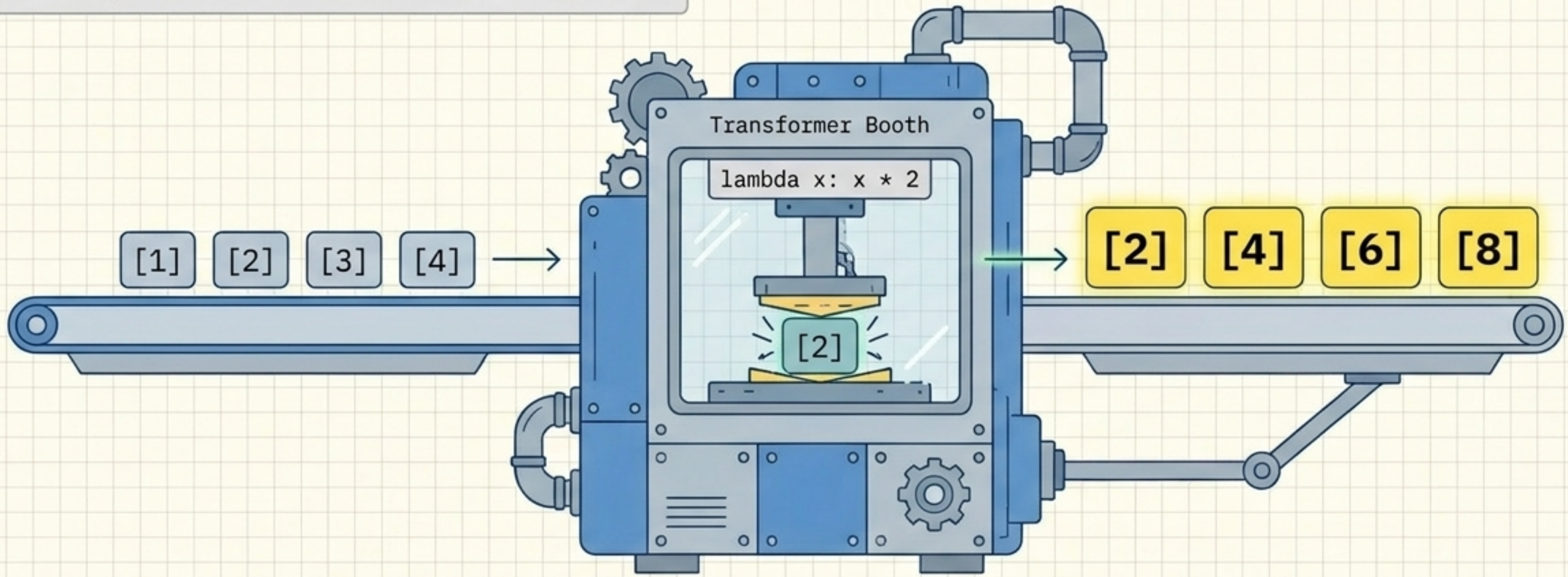
```
Code: filter(lambda x: x % 5 == 0, numbers)
```



ใช้กรองเอาเฉพาะข้อมูลใน List ที่ตรงตามเงื่อนไข (Expression เป็น True)

Visualizing **map()**: ตัวแปลงข้อมูล

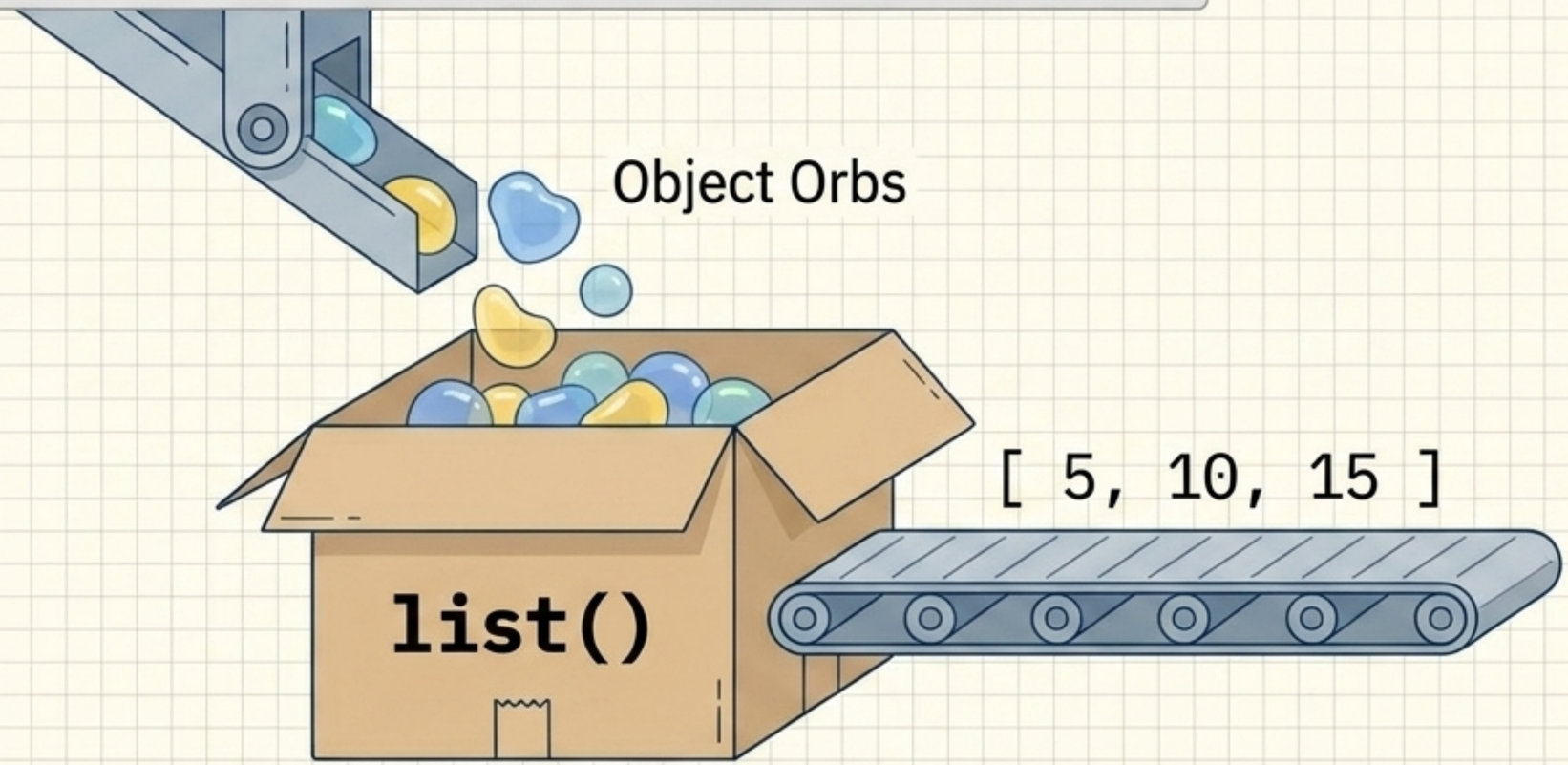
```
Code: map(lambda x: x * 2, numbers)
```



ใช้ปรับเปลี่ยนหรือคำนวณข้อมูล "ทุกๆ ตัว" ใน `List` ด้วยตรรกะเดียวกัน
แล้วส่งออกเป็นชุดข้อมูลใหม่ที่มีขนาดเท่าเดิม

Output Conversion: การแปลงกลับเป็น List

```
Code: result = list(filter(lambda x: x % 5 == 0, numbers))
```



map() และ filter() จะคืนค่ากลับมาเป็น "ออบเจ็ค (Object)" เพื่อประหยัดหน่วยความจำ เราต้องใช้ฟังก์ชัน list() ครอบอีกชั้นเพื่อแพ็คข้อมูลให้ออกมาเป็น List ที่อ่านได้ตามปกติ

Master Synthesis: เลือกใช้งานเครื่องมือให้ถูกจุด

Dimension Name	Regular Function (def)	Lambda Expression
การประกาศ (Declaration Keyword)	`def`	`lambda`
ชื่อ (Naming)	มีชื่อ (Named)	ไม่มีชื่อ (Anonymous)
ความยาว (Length)	หลายบรรทัด (Multi-line) โครงสร้างซับซ้อนได้	บรรทัดเดียว (Single Expression) เท่านั้น
การส่งค่า (Return)	ต้องเขียนคีย์เวิร์ด `return` ❌	ส่งค่ากลับอัตโนมัติ (Implicit) ✅
Use Case (เหมาะกับ)	โค้ดที่ต้องเรียกใช้ซ้ำหลายๆ ที่	ลอจิกแบบใช้ครั้งเดียวทิ้ง (Throwaway) หรือใช้คู่กับ map/filter

The Functional Data Pipeline: รีดโค้ดให้สั้น จัดการข้อมูลให้ทรงพลัง