การแก้ไขปัญหา
(Problem Solving)

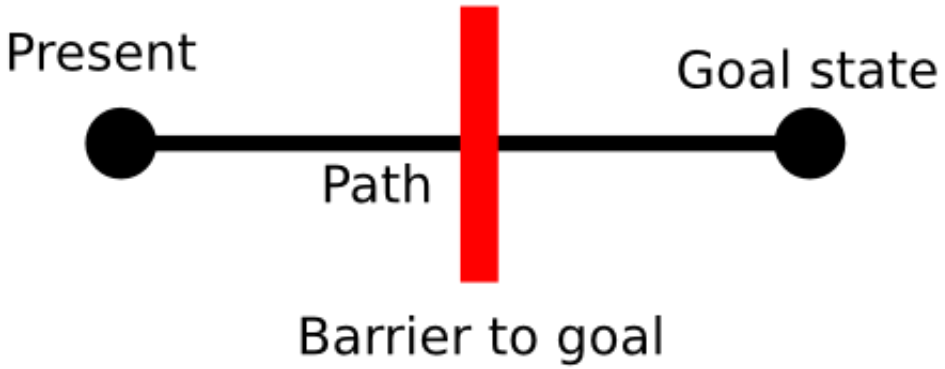
ฉัตรชัย เกษมทวีโชค

Chatchai.kase@ku.th

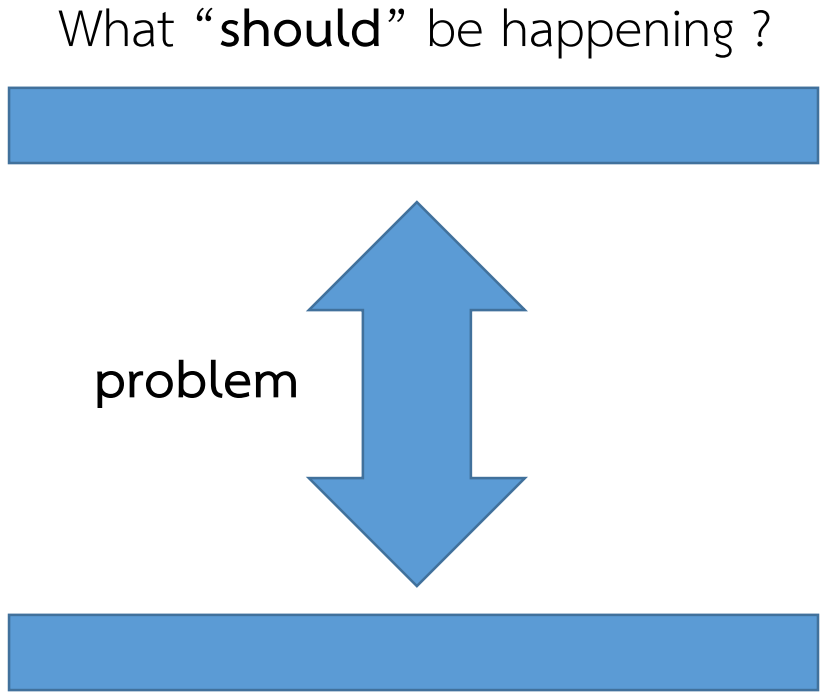
การแก้ไขปัญหา (Problem Solving)

- ประกอบด้วยการใช้วิธีทั่วไปหรือแบบเฉพาะกิจในลักษณะที่เป็นระเบียบเพื่อหาคำตอบสำหรับปัญหา
- บางส่วนของเทคนิคการแก้ปัญหาที่พัฒนาและใช้ในปัญญาประดิษฐ์วิทยาการคอมพิวเตอร์ วิศวกรรม คณิตศาสตร์หรือวิชาที่เกี่ยวข้องกับเทคนิคการแก้ปัญหาทางจิตในงานด้านจิตวิทยา

What is a problem?



Source: http://bearlamp.com.au/what-is-a-problem/



Source: Problem Solving & Root Cause Analysis: NMC Training & Research

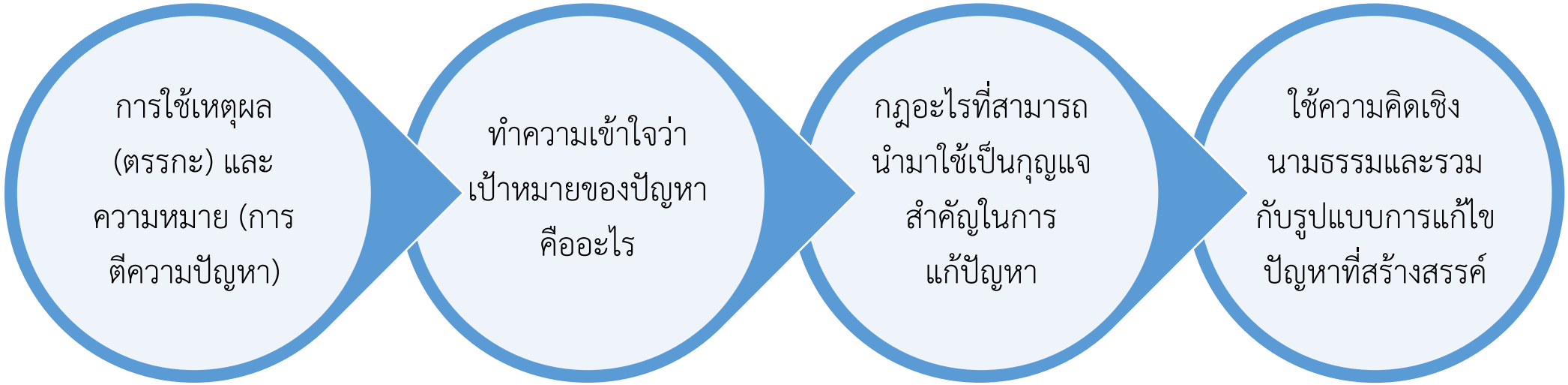
What is a problem?

1. อุปสรรคซึ่งทำให้ยากที่จะบรรลุเป้าหมายเป้าหมายหรือวัตถุประสงค์
2. สถานการณ์ สภาพหรือปัญหาที่ยังไม่ได้รับการแก้ไข
3. ในแง่เรื่องทั่วไป ปัญหาจะเกิดขึ้นเมื่อบุคคลตระหนักถึง
   - ความแตกต่างอย่างมีนัยสำคัญระหว่างสิ่งที่เป็นจริงและสิ่งที่ต้องการ

การแก้ไขปัญหา (Problem Solving)

1. Well-defined problem เป็นปัญหาที่มีวิธีการแก้ไขปัญหาที่ชัดเจน จึงเป็นปัญหาที่แก้ไขไม่ยากเท่าประเภทที่สอง
    1. มีเป้าหมายที่เฉพาะเจาะจง
    2. กำหนดเส้นทางการแก้ปัญหาที่ชัดเจน
    3. วางแผนวิธีการแก้ไขได้อย่างชัดเจน
2. Ill-defined problem เป็นปัญหาที่ไม่มีรูปแบบการแก้ไขที่ชัดเจนแน่นอน ต้องใช้ความคิดสร้างสรรค์

การแก้ไขปัญหา (Problem Solving)



https://americanwatercollege.org/creative-problem-solving-and-decision-making/

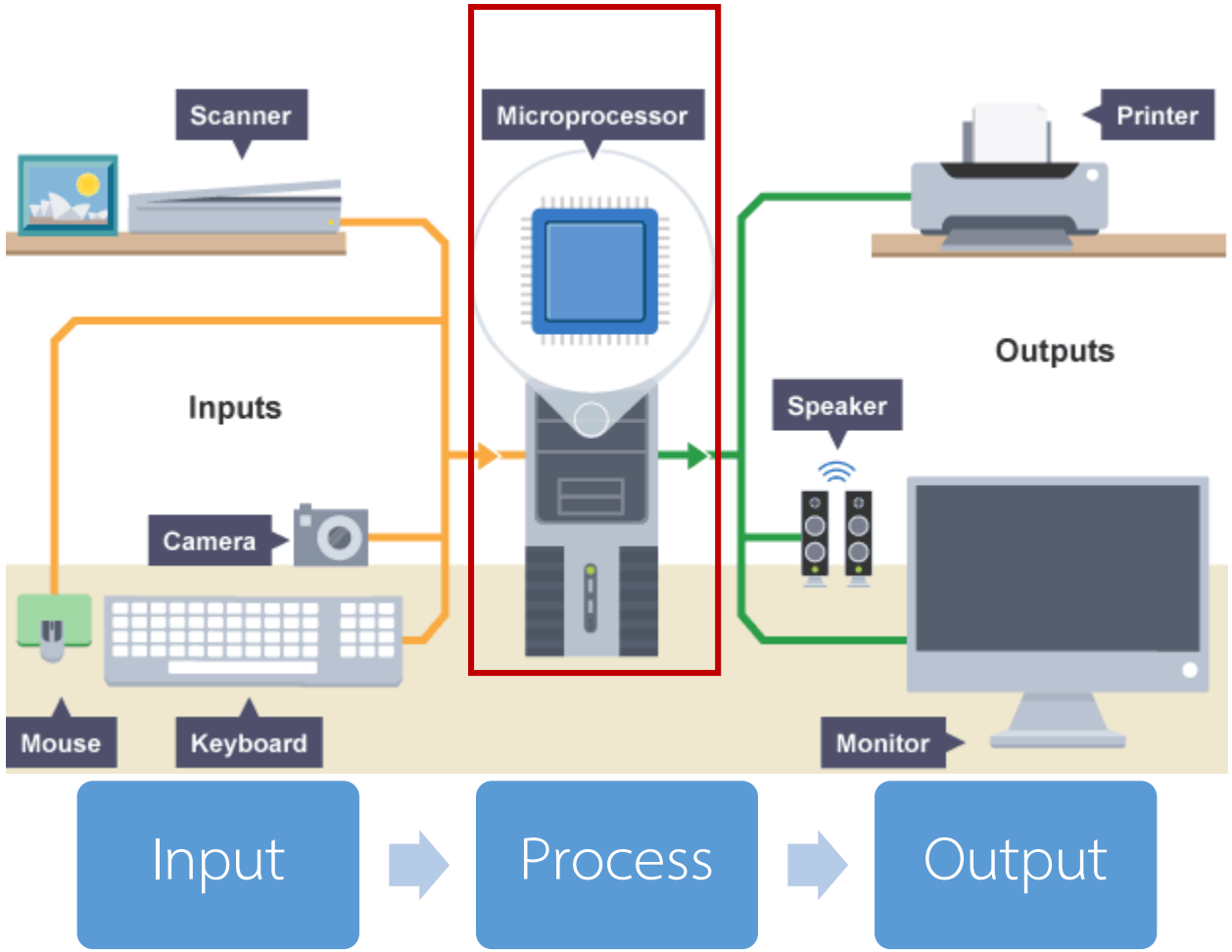
การค้นหาปัญหาหรือการวิเคราะห์ปัญหา → ค้นหาจุดเริ่มต้นและการดำเนินการแก้ปัญหา → การสร้างทางเลือกของกลยุทธ์ → การใช้งานและตรวจสอบกลยุทธ์ที่เลือก

(Problem Finding or Problem Analysis) → Problem shaping → Generating alternative strategies → Implementation and verification of the selected solution

Problem Solving (Computer Sciences)

- Problem Solving is the sequential process of analyzing information related to a given situation and generating appropriate response options.
- การแก้ไขปัญหาทางคอมพิวเตอร์คือขั้นตอนต่อเนื่องของการวิเคราะห์ข้อมูลที่เกี่ยวข้องกับสถานการณ์ที่กำหนดและสร้างตัวเลือกการตอบสนองที่เหมาะสม

Problem Solving (Computer Sciences

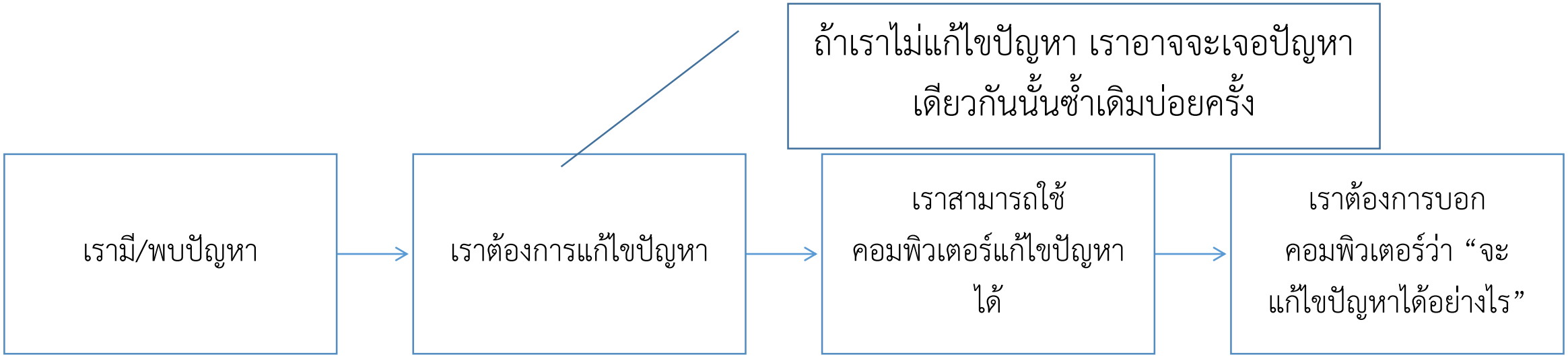


http://stangroundcomputing.co.uk

- วิทยาการคอมพิวเตอร์เป็นการศึกษาที่เกี่ยวกับการแก้ปัญหาด้วยคอมพิวเตอร์
- ปัญหาที่เราต้องการแก้ปัญหาอาจมาจากปัญหาจริงที่ปรากฏให้เห็นชัดเจน (รูปธรรม) หรือแม้แต่จากปัญหานามธรรม
- เราจำเป็นต้องมีแนวทางที่เป็นมาตรฐานในการแก้ปัญหา

ความจำเป็นต้องมีการเขียนโปรแกรม

ปัญหาจะแก้ไขได้โดยใช้คอมพิวเตอร์โดยได้รับข้อมูลการป้อนข้อมูลของผู้ใช้ (เช่นข้อมูลคีย์บอร์ด / เมาส์หรือการควบคุมเกม) จากนั้นประมวลผลการป้อนข้อมูลและการผลิตเอาต์พุตบางประเภท (เช่นภาพทดสอบเสียง) บางครั้งข้อมูลขาเข้าและขาออกอาจอยู่ในรูปแบบของฮาร์ดไดรฟ์หรืออุปกรณ์เครือข่าย

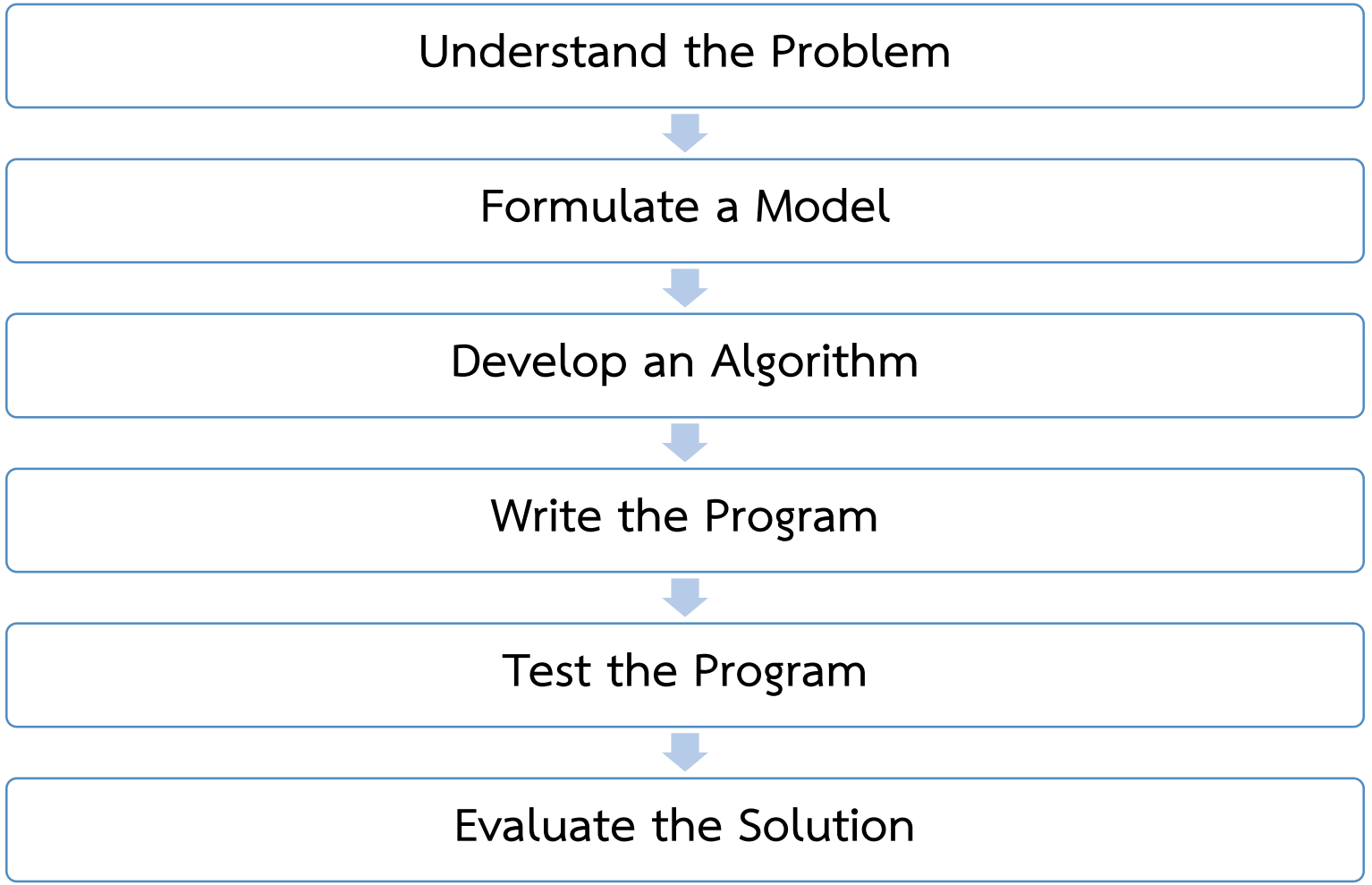


Source: Problem solving using Computers By : David Livingston J Sudesh Kantila

แนวทางการแก้ไขปัญหา

- เมื่อพบปัญหา พิจารณาว่าสามารถแก้ปัญหาด้วยการทำด้วยตัวเอง (Manual) หรือใช้คอมพิวเตอร์ได้
- พิจารณาขนาดของปัญหาและความซับซ้อนของปัญหา :
  - ถ้าปัญหาเล็กหรือง่ายที่สามารถแก้ด้วยตนเองหรือใช้คอมพิวเตอร์ ทำได้ทันที
  - ถ้าปัญหาใหญ่ขึ้นสามารถแบ่งย่อยออกเป็นปัญหาเล็ก ๆ (sub-problems) และเริ่มต้นแก้ปัญหาทีละขั้นตอน
    - หลังจากเสร็จสิ้นการแก้ปัญหาย่อยปัญหาใหญ่ทั้งหมดได้รับการแก้ไขได้ง่ายขึ้น

6 steps for solving problems



http://people.scs.carleton.ca/~lanthier/teaching/ProcessingNotes/COMP1405_Ch1_IntroductionToComputerScience.pdf

ขั้นตอนการพัฒนาโปรแกรม

1) กำหนดและวิเคราะห์ปัญหา

2) เขียนผังงานหรือซูโดโค้ด

3) เขียนโปรแกรม

4) ทดสอบและแก้ไขโปรแกรม

5) ทำเอกสารและบำรุงรักษาโปรแกรม

ตัวอย่าง

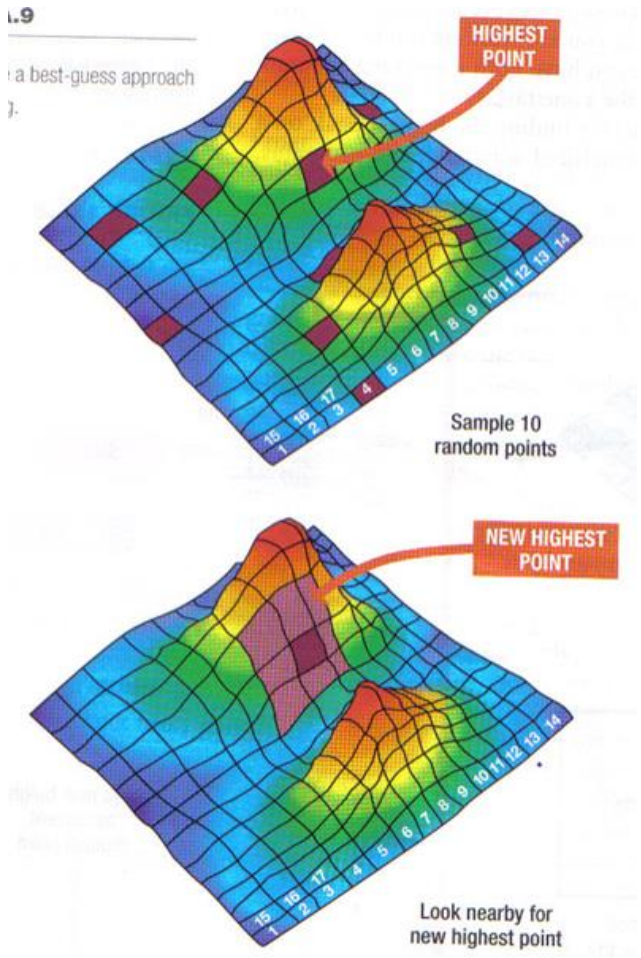
- การจัดรูปการแก้ไขปัญหาด้วยคอมพิวเตอร์เป็นแบบ input/process/output :
- ตัวอย่างการคำนวณค่าเฉลี่ยผลการเรียนของนิสิตทั้งหมดในชั้นเรียน

Input:	รับข้อมูลผลการเรียนทั้งหมดจากไฟล์ใน USB Drive หรือ อีเมล
Process:	1. หาผลรวมของผลการเรียนของนิสิตทั้งหมด 2. หาจำนวนนิสิตทั้งหมดว่ากี่คน 3. หาค่าเฉลี่ยของผลการเรียนทั้งหมด
Output:	แสดงผลลัพธ์ของค่าเฉลี่ยผ่านหน้าจอ เครื่องพิมพ์ และบันทึกเป็นไฟล์ใน Hard Disk หรือ USB Drive

อัลกอลิทึม (Algorithms)

- **อัลกอลิทึม (Algorithms)** : เป็นขั้นตอนการทำงานที่แสดงในแผนผัง IPO chart โดยรายการขั้นตอนดังกล่าว เราเรียกว่า algorithm.
- Algorithm เป็นลำดับขั้นตอนคำสั่งเรียงต่อเนื่องทีละคำสั่ง เพื่อให้เกิดการประมวลผลให้ได้ผลลัพธ์ตามต้องการ
- โดยที่ขั้นตอนเหล่านี้จะทำการค้นหาแนวทางในการแก้ไขปัญหา (solution) ด้วยเครื่องคอมพิวเตอร์หรือการทำงานด้วยมือ (manual or by hand) หรือใช้ทั้งสองอย่าง และลงมือเขียน flowchart เพื่อให้สำเร็จงานเดียวกัน

ฮิวริสทิค (Heuristics)



- **ฮิวริสทิค (Heuristics)** : คล้ายกับ algorithms ที่เป็นชุดรายการของขั้นตอนในการหาผลลัพธ์ของการแก้ไขปัญหา
- แต่ไม่เหมือน algorithms เพราะ heuristics จะเสนอหนทางที่ดีหรือเหมาะสมในการแก้ไขปัญหา ถึงแม้ว่าจะไม่ใช่วิธีที่ดีที่สุดก็ตาม

STEP 1: Understand the Problem

- ขั้นตอนแรกในการแก้ปัญหาก็เพื่อให้แน่ใจว่า เราเข้าใจปัญหาที่คุณกำลังพยายามแก้ไข
- สิ่งที่จำเป็นต้องรู้:
    1. ตอนนี้มีข้อมูล / ข้อมูลอะไรบ้าง ?
    2. ข้อมูลอยู่ในอะไร ?
    3. ข้อมูลมีรูปแบบอะไรบ้าง (data format) ?
    4. ข้อมูลอะไรที่หายไป (missing value) ?
    5. มีข้อมูลทุกอย่างที่ต้องการหรือไม่ ?
    6. ผลลัพธ์ที่ต้องการให้สร้างขึ้นมา (output) ?
    7. ลักษณะของผลลัพธ์ที่ต้องการเป็นแบบ ... ข้อความ รูปภาพ กราฟ ... ?
    8. สิ่งที่ฉันจะต้องคำนวณ ?

Data Examples

ลำดับ	รหัสประจำตัว	ชื่อ-สกุล	Total	SCORE	GRADE
1	571100001	นายกรกฎ งามเทียม	60	2.5	C+
2	571100002	นายกวิน กำลังทำ	51	2.0	C
3	571100003	นางสาวกิติรัตน์ บัวสวย	64	2.5	C+
4	571100004	นางสาวณัฐยา ศรีสุราษฎร์	-75	3.5	B+
5	571100005	นางสาวชนายา มาเสมอ	80	4.0	A
6	571100006	นางสาวทองสุก เอกลักษณ์		3.0	B
7	571100007	นางสาวทศพร สายงาม	74	3.5	B+
8	571100008	นายธนบัตร รวยมาก	61	2.5	C+
9	571100009	นางสาวพรภิยา พงษ์ศรีงาม	68	3.0	B
10	571100010	นางสาวบุตรี พลังมาก	81	4.0	A

- ตอนนี้มีข้อมูล / ข้อมูลอะไรบ้าง ?
- ข้อมูลอยู่ในอะไร ?
- ข้อมูลมีรูปแบบอะไรบ้าง (data format) ?
- ข้อมูลอะไรที่หายไป (missing value) ?
- ข้อมูลที่ผิดปกติ (Noise value) ?

STEP 2: Formulate a Model

- ปัญหาหลายอย่างแบ่งออกเป็นปัญหาเล็ก ๆ ที่ต้องมีการคำนวณทางคณิตศาสตร์แบบง่ายๆเพื่อประมวลผลข้อมูล
- ในตัวอย่างของเราเราจะคำนวณค่าเฉลี่ยของเกรดที่เข้ามา
  - ดังนั้นเราจำเป็นต้องทราบรูปแบบ (หรือสูตร) สำหรับการคำนวณค่าเฉลี่ยของพวงของตัวเลข
  - หากไม่มี "สูตร" ดังกล่าวเราจำเป็นต้องพัฒนาสูตรดังกล่าว
  - บ่อยครั้งที่ปัญหาแบ่งเป็นการคำนวณแบบง่ายๆที่เราเข้าใจดี
  - หากเราไม่แน่ใจว่าสูตรการคำนวณ เราสามารถค้นหาสูตรบางอย่างในหนังสือหรือทางออนไลน์ได้

Examples #1

สมมติให้คะแนนของนิสิตแต่ละคน เป็นตัวแปร $x_1, x_2, \ldots, x_n$

สูตรการหาค่าเฉลี่ย (Average) จะเท่ากับ

$$\text{Average1} = (x_1 + x_2 + x_3 + \ldots + x_n) / n$$

x_1	
x_2	
x_3	
x_4	
x_5	
x_6	
x_7	
x_8	
x_9	
x_{10}	
ค่าเฉลี่ย	

Examples #2

สมมติให้ค่าเกรดของนิสิตแต่ละคน เป็นตัวแปร $y_1, y_2, \ldots, y_n$

สูตรการหาค่าเฉลี่ย (Average) จะเท่ากับ

$$\text{Average2} = (y_1 + y_2 + y_3 + \ldots + y_n) / n$$

y_1	
y_2	
y_3	
y_4	
y_5	
y_6	
y_7	
y_8	
y_9	
y_{10}	
ค่าเฉลี่ย	

STEP 3: Develop an Algorithm:

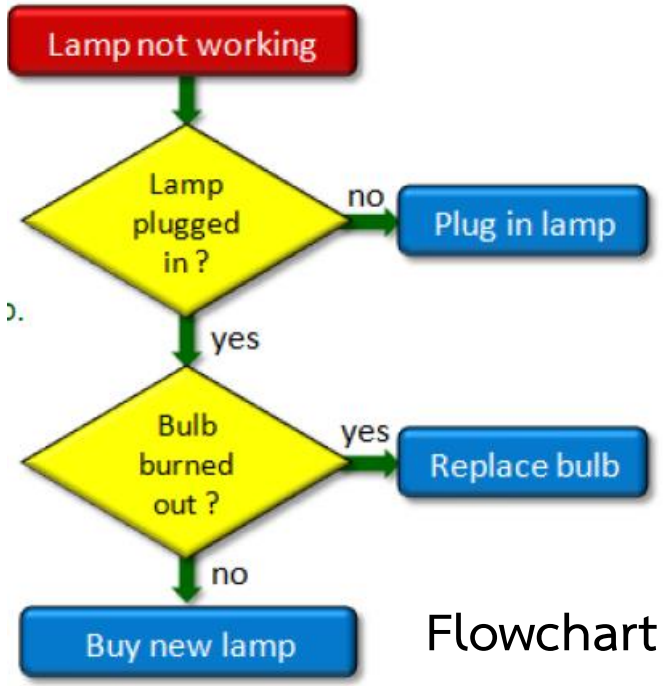
- ตอนนี้เราเข้าใจปัญหาและได้กำหนดรูปแบบแล้ว จะสามารถวางรูปแบบการประมวลผลได้ว่าเราต้องการให้คอมพิวเตอร์ทำอะไร หรือการสร้างอัลกอริทึม
- อัลกอริทึมคือลำดับของคำสั่งในการแก้ปัญหา (An algorithm is a precise sequence of instructions for solving a problem.)
- อัลกอริทึมที่ซับซ้อนมากขึ้นอาจถือเป็น "อัลกอริธึมแบบสุ่ม (randomized algorithms)" หรือ "อัลกอริธึมเชิงไม่กำหนด (non-deterministic algorithms)" ซึ่งคำสั่งไม่จำเป็นต้องเป็นลำดับและในอาจไม่มีจำนวนจำกัดความของคำสั่งที่ใช้
- https://www.youtube.com/watch?v=iJmMw0MpaKY
- https://www.youtube.com/watch?v=_vhSu4xBoFs

STEP 3: Develop an Algorithm:

- เพื่อพัฒนาอัลกอริทึม เราจำเป็นต้องแสดงคำสั่งที่จะต้องใช้ในรูปแบบให้กับบุคคลที่เกี่ยวข้องที่เข้าใจได้
- สองวิธีการที่ใช้งานโดยทั่วไปคือ
  1. pseudo code เป็นลำดับคำสั่งภาษาอังกฤษที่เรียบง่ายและรัดกุมของเพื่อแก้ไขปัญหา
  2. flow charts เป็นการแสดงกราฟิกสัญลักษณ์แต่ละขั้นตอนการทำงาน

1. IF lamp works, go to step 7.
2. Check if lamp is plugged in.
3. IF not plugged in, plug in lamp.
4. Check if bulb is burnt out.
5. IF blub is burnt, replace bulb.
6. IF lamp doesn’t work, buy new lamp.
7. Quit ... problem is solved.

Pseudo code



Flowchart

STEP 3: Develop an Algorithm:

Pseudocode	Flowchart
เหมาะสมกับการอธิบายโปรแกรมให้คนที่**ไม่**รู้การเขียนโปรแกรม	เป็นกราฟิกที่สามารถเห็นภาพการทำงานทั้งหมดของโปรแกรมได้อย่างรวดเร็ว
เหมาะสมกับการเขียนในเอกสารกระดาษ	สามารถใช้รูปทรงและสีของกราฟิกในการแบ่งแยกประเภทการทำงาน ดูสวยและสะดุดตามากกว่า
นำไปเขียนเป็นคำสั่งโปรแกรมคอมพิวเตอร์ได้ง่าย	ลดข้อจำกัดทางภาษาได้มาก

STEP 3: Develop an Algorithm:

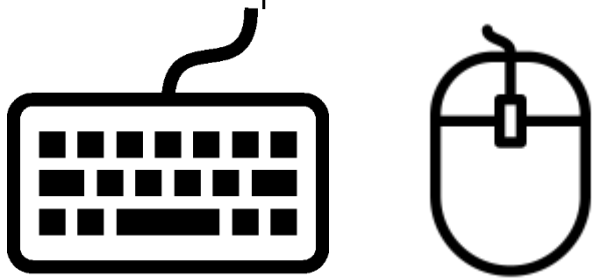
- รูปแบบคำสั่งที่ใช้ในโปรแกรมคอมพิวเตอร์แบ่งออกเป็น 7 แบบคือ

1. **อ่านหรือนำเข้าข้อมูล (Read / Input Data) :** คำสั่งที่ใช้ในการอ่านข้อมูลจากคีย์บอร์ด ไฟล์ข้อมูลภายในเครื่อง (Local computer) เครื่องแม่ข่าย (Server) หรืออินเตอร์เน็ต
2. **เรียงลำดับ (Sequence) :** ชุดคำสั่งที่ประกอบด้วยคำสั่งที่ให้คอมพิวเตอร์ทำงานแบบเรียงลำดับจากบนลงล่าง
3. **เงื่อนไข (Condition) :** ชุดคำสั่งที่ให้คอมพิวเตอร์เลือกทำงานอย่างใดอย่างหนึ่งตามเงื่อนไขที่กำหนดไว้
4. **วนซ้ำ (Repetition, Iteration) :** ชุดคำสั่งที่ให้คอมพิวเตอร์ทำงานคำสั่งชุดใดชุดหนึ่งซ้ำกันหลายครั้งตามเงื่อนไขที่กำหนดไว้
5. **ข้ามคำสั่ง (Jumping) :** คำสั่งที่ใช้ในคอมพิวเตอร์ข้ามไปประมวลผลหรือทำงานที่ตำแหน่งบรรทัดคำสั่งที่กำหนดไว้
6. **เขียน/บันทึกข้อมูล (Write / Save Data) :** คำสั่งบันทึกผลการทำงานหรือสารสนเทศที่ได้ลงไปจัดเก็บในแฟ้มข้อมูล
7. **แสดงผลลัพธ์ (Print/Display) :** คำสั่งที่ใช้ในการแสดงผลลัพธ์ออกทางหน้าจอคอมพิวเตอร์หรือเครื่องพิมพ์

แผนผังการทำงาน

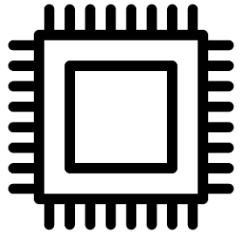
Input

- Read / Input Data



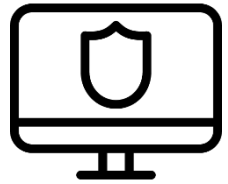
Process

- Sequence
- Condition
- Repetition



Output

- Print / Display Data



Storage

- Write / Save Data

Source: http://www.iconsmind.com

http://www.customicondesign.com

Example

Sequence

1. Make sure switch is turned on.
2. Check if lamp is plugged in.
3. Check if bulb is burnt out.

...

Condition

If lamp is not plugged in.

    then plug it in.

If bulb is burned out.

    then replace bulb.

Otherwise buy new lamp.

Repetition

Repeat

    get a new light bulb.

    put it the lamp.

Until lamp works or no more bulb.

For i = 1 to 3

    get a new light bulb.

    put it the lamp.

next

jumping

If bulb works

    then **goto** step 7

STEP 4: Write the Program:

- ตอนนี้มีขั้นตอนในการแก้ปัญหาได้อย่างถูกต้องแล้ว ถึงขั้นตอนการแปลงอัลกอริทึมจากขั้นตอนที่ 3 เป็นชุดคำสั่งที่คอมพิวเตอร์สามารถเข้าใจได้
- การเขียนโปรแกรมมักจะเรียกว่า "writing code" หรือ "implementation a algorithm" ดังนั้นชุดคำสั่ง (source code หรือซอร์สโค้ด) เป็นตัวโปรแกรมที่ใช้สั่งงานคอมพิวเตอร์
- อัลกอริธึมของเราในการหาค่าเฉลี่ยของชุดของเกรดมีลักษณะคล้ายกันกับชุดคำสั่งในเชิงโครงสร้าง

Pseudocode	Processing code (i.e., program)

1. set the sum of the grade values to 0.
2. load all grades x_1 ... x_n from file.
3. **repeat n times** {
4. get grade x_i
5. add x_i to the sum
}
6. compute the average to be sum / **n**.
7. print the average.
	`int sum = 0;`
	`byte[] x = loadBytes("numbers");`
	`for (int i=0; i<x.length; i++)`
	`sum = sum + x[i];`
	
	`int avg = sum / x.length;`
	`print(avg);`

STEP 4: Write the Program:

- ในความเป็นจริง ชุดคำสั่งที่เขียนจะแตกต่างกันขึ้นอยู่กับภาษาโปรแกรมที่ใช้
- การเรียนรู้ภาษาเขียนโปรแกรมอาจดูเหมือนยากในตอนแรก แต่จะง่ายขึ้นเมื่อมีการฝึกปฏิบัติต่อเนื่อง
- **ความยากในการเขียนโปรแกรม :** คอมพิวเตอร์ต้องการการเขียนคำสั่งโปรแกรมที่ถูกต้องและแม่นยำ เพื่อให้คอมพิวเตอร์เข้าใจถึงสิ่งที่คุณต้องการให้ทำ

คนเขียนโปรแกรมต้องจำคำสั่งให้ขึ้นใจ สามารถเขียนคำสั่งได้ทันที

การฝึกฝนบ่อย ๆ ทำให้เกิดความชำนาญคุ้นเคย

- ถ้าเขียนโปรแกรมที่ผิดจากกฎเกณฑ์ของภาษา จะทำให้โปรแกรมของคุณเกิด**ข้อผิดพลาดในการคอมไพล์** (Compile Errors)
- **Compile** คือกระบวนการแปลงคำสั่งโปรแกรมให้เป็นชุดคำสั่งภาษาเครื่องที่คอมพิวเตอร์สามารถเข้าใจได้

STEP 5: Test the Program:

- เมื่อคุณมีโปรแกรมเขียนที่คอมไพล์แล้วคุณต้องตรวจสอบให้แน่ใจว่าได้แก้ไขแล้ว
- โปรแกรมจะแก้ปัญหาได้อย่างถูกต้องด้วย Running program ซึ่งคือการเรียกใช้โปรแกรมคือการบอกคอมพิวเตอร์เพื่อทำงานตามคำสั่งที่คอมไพล์แล้ว
- บางครั้งโปรแกรมของคุณทำงานได้อย่างถูกต้องสำหรับชุดข้อมูลป้อนข้อมูลบางส่วน แต่ไม่ครบทุกกรณีปัญหาหรือข้อมูลทั้งหมด
  - อาจเป็นไปได้ว่าคุณไม่ได้สร้างขั้นตอนวิธีที่เหมาะสม เพื่อจัดการกับสถานการณ์ทั้งหมดที่อาจเกิดขึ้น ดังนั้นอาจค้นหาคำสั่งที่ไม่ถูกต้องแล้วแก้ไขคำสั่งนั้นจากลำดับ
  - สิ่งที่เกิดขึ้นปัญหาดังกล่าวกับโปรแกรมของคุณเรียกว่าข้อบกพร่อง หรือ bug
- Bugs คือปัญหา / ข้อผิดพลาดที่เกิดขึ้นกับโปรแกรมที่ทำให้มันหยุดทำงานหรือผลลัพธ์ที่ไม่ถูกต้องหรือไม่พึงประสงค์
  - เพื่อหาข้อผิดพลาดได้อย่างมีประสิทธิภาพ จึงควรทดสอบโปรแกรมด้วย test case หลายรายการ
  - กระบวนการในการค้นหาและแก้ไขข้อผิดพลาดในโค้ดของคุณเรียกว่า การดีบัก (Debug) และขั้นตอนนี้จะใช้เวลามาก
- กระบวนการออกแบบและเขียนโปรแกรมที่ครบถ้วนและรอบคอบ จะช่วยลดจำนวนบักในโปรแกรม และควรจะแก้จุดบกพร่องได้ง่ายขึ้น

STEP 6: Evaluate the Solution:

- เมื่อโปรแกรมของคุณสร้างผลลัพธ์ที่ดูเหมือนถูกต้องแล้วคุณต้องพิจารณาปัญหาดั้งเดิมอีกครั้งและตรวจสอบให้แน่ใจว่าคำตอบถูกจัดรูปแบบให้เป็นทางออกที่เหมาะสมกับปัญหา
- บ่อยครั้งจะพบว่าผลลัพธ์ของโปรแกรมไม่สามารถแก้ปัญหาตามที่ต้องการได้ คุณอาจทราบว่ามีขั้นตอนเพิ่มเติมเกี่ยวข้อง
  - คุณต้องการข้อมูลเพิ่มเติมเพื่อแก้ปัญหาได้อย่างเต็มที่ หรือ
  - บางทีคุณจำเป็นต้องปรับเปลี่ยนผลลัพธ์เพื่อแก้ปัญหาได้อย่างมีประสิทธิภาพ (เช่นเกมของคุณช้าเกินไป)
- สิ่งสำคัญคือต้องจำไว้ว่าคอมพิวเตอร์จะทำในสิ่งที่คุณบอกให้ทำเท่านั้น ขึ้นอยู่กับคุณในการตีความผลลัพธ์ในลักษณะที่มีความหมายและกำหนดว่าจะแก้ปัญหาเดิมหรือไม่
  - เราอาจจำเป็นต้องทำอีกบางขั้นตอนอีกครั้งบางทีอาจจะย้อนกลับไปในขั้นตอนที่ 1 อีกครั้งหากข้อมูลหายไป