Mathematics for Working Programmers: Day 0

by Dave Rawitat Pulam

“คณิตศาสตร์” คืออะไร แล้วทำไมถึงเป็นสิ่งที่สำคัญมากกับการวิเคราะห์และออกแบบระบบซอฟต์แวร์ ไม่ว่าจะเป็นการออกแบบ Abstraction, Architecture หรือ Algorithm ไม่ใช่แค่เพียงการนำไปใช้ในการคำนวน หรือสร้างสูตรคำนวนต่างๆ เท่านั้น

ซึ่งเรื่องนี้เราคงพอจะเห็นมาบ้าง เวลาที่อ่านหนังสือ Data Structures หนังสือ Algorithm หรือหนังสือทฤษฎีการคำนวน (Theory of Computation) ที่อธิบายทุกอย่างด้วยแบบจำลองทางคณิตศาสตร์ ไม่ใช่อธิบายด้วยโค้ดของโปรแกรม ซึ่งโดยทั่วไปเราจะมีปัญหาค่อนข้างมากกับการอ่านและทำความเข้าใจหนังสือเหล่านี้

คลาสนี้ถือได้ว่าเป็นการปูพื้นฐานทางคณิตศาสตร์ที่จำเป็น เพื่อนำไปพัฒนาและปรับกระบวนทัศน์ (Thought Paradigm) และสร้างกรอบความคิด (Thinking Framework) และพัฒนาการหยั่งรู้ในระดับสามัญสำนึก (Developing Intuition) ว่าคณิตศาสตร์นั้นคืออะไร อยู่ในที่ไหน มีอะไรบ้างรอบตัวเรา และในซอฟต์แวร์ที่เรากำลังพัฒนา หรือกำลังจะต้องพัฒนา แม้ว่าเรื่องเหล่านี้จะไม่มีการคำนวน ไม่มีตัวเลขเลยก็ตาม

เนื้อหาของคอร์ส

วันที่ 1: Reasoning with Discrete Mathematics

Session 1: Misconceptions about Mathematics

- General misconceptions about Mathematics
- Calculations, Computations, Algorithms, Programs, and Mathematics

Session 2: Logical Studies of Logical Things

- Reintroduction to Set, Logic, and Functions
- Reintroduction to Logical Structures

วันที่ 2: Logical and Computational Structures

Session 1: Modeling & Reasoning on Logical Structures

- Modeling everyday life scenarios with simple Logical Structures
- Logical structures, Problem Solving, and deriving Algorithms.
- Nature of Logical Structures: List, Tree, Graphs.
- Recursive Structures and Recursive Reasoning.

Session 2: Logical Structures as Model of Computation

- Introduction to Computability and Computational Complexity
- Introduction to Information Theory and Information Entropy

(หมายเหตุ: เนื้อหาบางส่วนอาจมีการปรับออก ตามความเหมาะสมของเวลาในคอร์ส)

Mathematics for Working Programmers: Day 1 & 2

by Dave Rawitat Pulam

“Functional Programming” (จากนี้จะย่อว่า **“FP”**) ถือเป็นรูปแบบการคิดแก้ปัญหา และการเขียนโปรแกรมแบบหนึ่ง ที่แม้จะเก่าแก่ไม่แพ้ Imperative Programming (ซึ่งเป็นรูปแบบที่เราส่วนมากคุ้นเคยกัน) แต่ปฏิเสธไม่ได้ว่า FP นั้นไม่เคยที่จะเป็นแนวการคิดและเขียนโปรแกรมที่ใช้งานอย่างแพร่หลายแต่อย่างใด จะมีอยู่ก็แต่ในวงแคบเท่านั้น

จนกระทั่งในช่วงทศวรรษที่ผ่านมานี้ ได้มีการนำ FP ไม่ว่าจะเป็นภาษาโปรแกรมที่เป็น FP โดยตรง หรือการนำเอาแนวคิดพื้นฐานและคอนเซปท์ต่างๆ ของ FP มาใช้ในการพัฒนาโปรแกรมมากขึ้น เราเริ่มได้ยินศัพท์เช่น Functors และ Monads เราเริ่มเจอโจทย์ที่เหมาะกับการคิดแบบ Highly-Distributable ของ FP มากขึ้นเรื่อยๆ รวมถึงกระบวนการพัฒนาโปรแกรมที่ถือว่าเป็น Best Practice ก็ยังมีแนวคิดแบบ FP อยู่เยอะมาก

แต่เนื่องจากรากฐานของความคิดของ FP นั้นมากจาก “แบบจำลองคณิตศาสตร์สำหรับการคำนวนฟังก์ชันแบบ Pattern Matching & Term Rewriting” ที่รู้จักกันในชื่อ Lambda Calculus ซึ่งแตกต่างจาก Imperative Programming ที่เราคุ้นเคยกัน อันมีรากฐานมาจาก “แบบจำลองคณิตศาสตร์สำหรับการคำนวนแบบ Instruction-based, State & Memory Centric” ซึ่งกลายมาเป็นแบบจำลองของเครื่องจักรคำนวนที่เรารู้จักกันในชื่อ Turing Machine ทำให้ผู้เขียนโปรแกรมอาจต้องปรับเปลี่ยนวิธีคิด วิธีการในการแก้ไขปัญหา ถึงจะเข้าใจและใช้ FP ได้อย่างที่ควรจะเป็น

คอร์สนี้มุ่งเน้นการสอน **“พื้นฐานความคิดแบบ Functional ซึ่งมีรากฐานจากคณิตศาสตร์”** สำหรับการแก้ปัญหาต่างๆ โดยใช้พื้นฐานของ Set และ Function ซึ่งนำไปสู่โค้ดและโปรแกรมที่ Readable, Reasonable, Maintainable, Testable, Distributable, Safe และมีการอภิปรายถึงการนำแนวคิดดังกล่าวในการออกแบบ Test-Case และ Architecture ของระบบที่มีขนาดใหญ่ขึ้น

โดยมี **“การแสดงตัวอย่าง โดยการใช้ภาษา Haskell”** ซึ่งเป็นภาษาโปรแกรมแบบ Purely Functional และเป็นต้นแบบของเครื่องมือหลายตัวในภาษาอื่นๆ หลายภาษาในปัจจุบัน ซึ่งแม้ว่าการแสดงตัวอย่างจะใช้ภาษา Haskell แต่พื้นฐานความคิดและวิธีการใช้แก้ปัญหานั้น สามารถใช้กับภาษาโปรแกรมไหนก็ได้

เนื้อหาของคอร์ส

วันที่ 1

Session 1: Introduction to modern computer programming paradigms

- Retrospective of programming language landscape in the pass decade.
- The world had been slowly moving toward new paradigm of programming, and the keyword is “Type”: Statically Typed, Strong Type-Checking, Generic Programming with Parametric Types, Type Constraint, Domain Modeling with Type, etc. What is the cause of these changes? How should we understand them so we could use them effectively?

Session 2: Theory of Computation and Problem-Solving Paradigms

- Problem, Problem-Solving, Computation, and Algorithm
- Theory and Models of Computation
- Foundation of Programming Paradigms
- Turing Machine: The Theoretical Model for Imperative Paradigm

วันที่ 2

Session 1: Functional Programming Paradigm

- Lambda Calculus: The Theoretical Model for Functional Paradigm
- Problem Solving with Mathematical Functions (Pure Functions).
- Functional Programming and its relation to modern software development methodologies. I.e., how Functional Programming leads to more easily testable, readable, maintainable, distributable and purposeful codes.

Session 2: Programming in Type-Centric, Purely Functional Paradigm.

- Values, Containers (List, Tuple), Basic of Functions (Function Types, Function Composition, Currying), Variables, Functions and Patterns, Functional & Declarative Programming, Computation by Pure Function, Pattern Matching, and Term-Rewriting, Higher-Order Functions, Defining and Writing Functions, List Comprehension, Recursion and Induction, Custom Types

(หมายเหตุ: เนื้อหาบางส่วนอาจมีการปรับออก ตามความเหมาะสมของเวลาในคอร์ส)

Mathematics for Working Programmers: Day 3 & 4

by Dave Rawitat Pulam

หมายเหตุ: คลาสนี้เป็นคลาสต่อเนื่องจาก Mathematics for Working Programmers: Day 1 & 2 ผู้ประสงค์ที่จะเรียนคลาสนี้ จะต้องเคยเรียนคลาส Day 1 & 2 มาก่อน เท่านั้น

ในคลาส Mathematics for Working Programmers: Day 1 & 2 นั้น เราได้ทำความรู้จักกับ Functional Programming Paradigm ซึ่งรวมไปถึงการคิดออกแบบโปรแกรมแบบ Functional และเครื่องมือต่างๆ ที่มีใน Functional Paradigm ที่กำลังค่อยๆ ถูกทะยอย implement ลงไปในภาษา mainstream ที่โดยมากมีพื้นฐานเป็น Imperative Programming Language มากขึ้นเรื่อยๆ

ในคลาสนี้ เราจะไปพบกับ step ต่อไปในเส้นทางการคิดแก้ปัญหาแบบ Functional ซึ่งก็คือ “การแก้ปัญหาด้วยการพิสูจน์และการหาเหตุผลแบบคณิตศาสตร์” โดยใช้วิธี Equational Reasoning และ Mathematical Induction (โดยเฉพาะอย่างยิ่งการใช้งาน Induction เป็นเครื่องมือในการ reasoning กับ Recursive Structures) เป็นหลักในการพิสูจน์คุณสมบัติต่างๆ ของฟังก์ชันและ component ต่างๆ ที่สามารถนำมาใช้ได้ในการแก้ปัญหา จนกลายมาเป็น Functional Law ต่างๆ ที่สามารถนำไปใช้ได้

ซึ่งการพิสูจน์นี้จะนำเราไปสู่ Functional Expression, Data Structure, และ Algorithm ที่ตรงกับปัญหาที่ต้องการแก้ มีประสิทธิภาพ และพิสูจน์ความถูกต้องได้มากยิ่งขึ้น ผ่านตัวอย่างที่เห็นได้ชัดเจนเป็นรูปธรรม โดยในคลาสนี้ เราจะทำการ optimize algorithm ในการแก้ปัญหาบางตัว จาก Naive Solution ที่เป็น *O(n^3)* ไปหา *O(n^2)* และสุดท้ายได้ optimal algorithm (สำหรับปัญหาที่ยกมา) ที่ *O(n)* โดยใช้วิธีการ Equational Reasoning และประยุกต์ใช้ Functional Law ที่ค้นพบมาแล้ว นอกจากนี้แล้ว เรายังจะเห็นการประยุกต์ใช้ Functional Law เหล่านี้ ในการช่วยคิดและออกแบบ architecture, data pipeline ที่มีประสิทธิภาพมากขึ้น มีคอขวดน้อยลง และสามารถพิสูจน์ความถูกต้องได้

ไม่เพียงเท่านั้น เราจะยังได้ทำความเข้าใจกับ Generic Type, Parametric Type และ Type Constraint ในระดับที่ลึกยิ่งขึ้นไปอีกด้วย

ท้ายที่สุด เราจะได้ทำความรู้จักกับ Category Theory ทั้งในฐานะของเครื่องมือในการคิดและออกแบบ abstraction ของโปรแกรม จนถึงเครื่องมือในการพัฒนาโปรแกรม ซึ่งเราจะได้เห็นว่า ทำไม programming techniques หลายตัวในปัจจุบัน ถึงมีรากฐานความคิดจาก Category Theory นี้มากขึ้นเรื่อยๆ ไม่ว่าจะเป็น Functor ประเภทต่างๆ, Monoid, Monad เป็นต้น

โดยมี **“การแสดงตัวอย่าง โดยการใช้ภาษา Haskell”** ซึ่งเป็นภาษาโปรแกรมแบบ Purely Functional และเป็นต้นแบบของเครื่องมือหลายตัวในภาษาอื่นๆ หลายภาษาในปัจจุบัน ซึ่งแม้ว่าการแสดงตัวอย่างจะใช้ภาษา Haskell แต่พื้นฐานความคิดและวิธีการใช้แก้ปัญหานั้น สามารถใช้กับภาษาโปรแกรมไหนก็ได้

เนื้อหาของคอร์ส

วันที่ 1

Session 1: Program Design with Types

- Problem, Problem Solving, General Recursive Function, and Recursively Enumerable Paradigm.
- Equivalency (and whats not) of Sets and Types. Algebra of Sets and Algebra of Types.
- Introduction to Generic Design with Type Constraints.

Session 2: Recursive Structures and Recursive Types

- Definition and Examples.
- Reasoning on Recursive Structures with Mathematical Induction.
- Algorithms on Recursive Structures.
- Proving Properties of Functions, Algorithms, on Recursive Structures.
- Generalized Recursion on Recursive Structures.

วันที่ 2

Session 1: Equational Reasoning with Functional Laws

- Equational Reasoning.
- Transforming Expressions using Functional Laws.
- Designing Scalable Architecture using Functional Design, based on Functional Laws.
- Optimizing Functional Algorithms with Equational Reasoning, Functional Laws, Mathematical Proofs, etc.

Session 2: Introduction to Category Theory as Abstraction Design Tool

- Introduction to concepts from Category Theory: Category, Functors, Monoids, Monads.
- Example of abstractions designed with Category: Domain Modeling, Adaptable Data Pipeline, Software/System Architecture, etc.

(หมายเหตุ: เนื้อหาบางส่วนอาจมีการปรับออก ตามความเหมาะสมของเวลาในคอร์ส)

Mathematics for Working Programmers: Day 5 & 6

by Dave Rawitat Pulam

หมายเหตุ: คลาสนี้เป็นคลาสต่อเนื่องจาก Mathematics for Working Programmers: Day 3 & 4 ผู้ประสงค์ที่จะเรียนคลาสนี้ จะต้องเคยเรียนคลาส Day 3 & 4 มาก่อน เท่านั้น

ในคลาส Mathematics for Working Programmers: Day 3 & 4 นั้น เราจบด้วยการทำความรู้จักกับ Category Theory ในฐานะเครื่องมือการคิดและออกแบบ Abstraction ตลอดจนทำความรู้จักกับเครื่องมือสำคัญคือ Functor ในแบบคร่าวๆ

ในคลาสนี้เราจะไปทำความรู้จักและทำความเข้าใจ Category Theory กันมากขึ้น เริ่มจากความคิดพื้นฐาน นิยามต่างๆ ในทางคณิตศาสตร์ ซึ่งจะต่อไปยังเครื่องมือต่างๆ ที่มีให้ใช้ในภาษาโปรแกรมในปัจจุบัน (บางตัวอาจจะยังไม่แพร่หลายถึงขนาดที่มีในทุกภาษาโปรแกรม แต่ก็ค่อยๆ มีมากขึ้นเรื่อยๆ) เช่น Functor, Monoid, Monad

เนื้อหาสำคัญก็คือ การนำพื้นฐานความคิดจาก Category Theory และระบบ Type System ใน Functional Programming มาใช้ในการออกแบบสถาปัตยกรรม โครงสร้างของระบบ และกำหนด Abstraction ต่างๆ เพื่อให้ซอฟต์แวร์มี Abstraction ที่สะท้อนความเป็นเหตุเป็นผลตามแต่ละ Domain ให้ได้มากที่สุด

เนื้อหาของคอร์ส

วันที่ 1

Session 1: Category Theory

- Formal Definition & Intuition
- Category: Objects & Morphisms.
- Category as Domain.
- Domain Reasoning with Category.
- Domain-Centric Algorithm Construction.

Session 2: Functors: Mapping between Categories.

- Formal Definition & Intuition
- Higher-Kind Types and Generic Types.
- Various kinds of Functors (Covariant Functor, Bifunctor, Contravariant Function, Profunctor).

วันที่ 2

Session 1: Why Functor Matters?

- Structure-Preserved Mapping. Across-Domain Reasoning.
- Designing Software Architecture and Abstractions with Category and Functors.

Session 2: Natural Transformation, Monoid, and Monad

- Formal Definition & Intuition

- Contextualization.
- Categorical Understanding of “Theory & Practice” as “Generic Core & Domain Specific Shell” in various applications, especially software development.
- Computational Effects as Monads.
- Solving Contextual Composition Problem with Monadic Composition.

(หมายเหตุ: เนื้อหาบางส่วนอาจมีการปรับออก ตามความเหมาะสมของเวลาในคอร์ส)

ทุกคลาสสอนโดย

อ. รวิทัต ภู่หลำ (อ. เดฟ)

Chief Software Architect & Managing Director, Code App Co., Ltd.

B. Sc, M. Sc Information Science and Engineering (University of Tsukuba, ญี่ปุ่น)

นักคณิตศาสตร์และวิศวกรซอฟต์แวร์ ที่เขียนโปรแกรมมากว่า 30 ปี และเขียน Functional Programming มากว่า 20 ปี (เริ่มต้นด้วย Haskell ตั้งแต่เรียนมหาวิทยาลัย) อดีตนักเรียนทุนรัฐบาลและอดีตอาจารย์มหาวิทยาลัย