Curry-Howard Isomorphism: Understanding the Connection Between Logic, Programming, and Polymorphism

This presentation explores the elegant correspondence between mathematical logic and computer programming known as the Curry-Howard isomorphism.

Yu **by Yu Feng**

What is Curry-Howard Isomorphism?

1 A Profound Connection

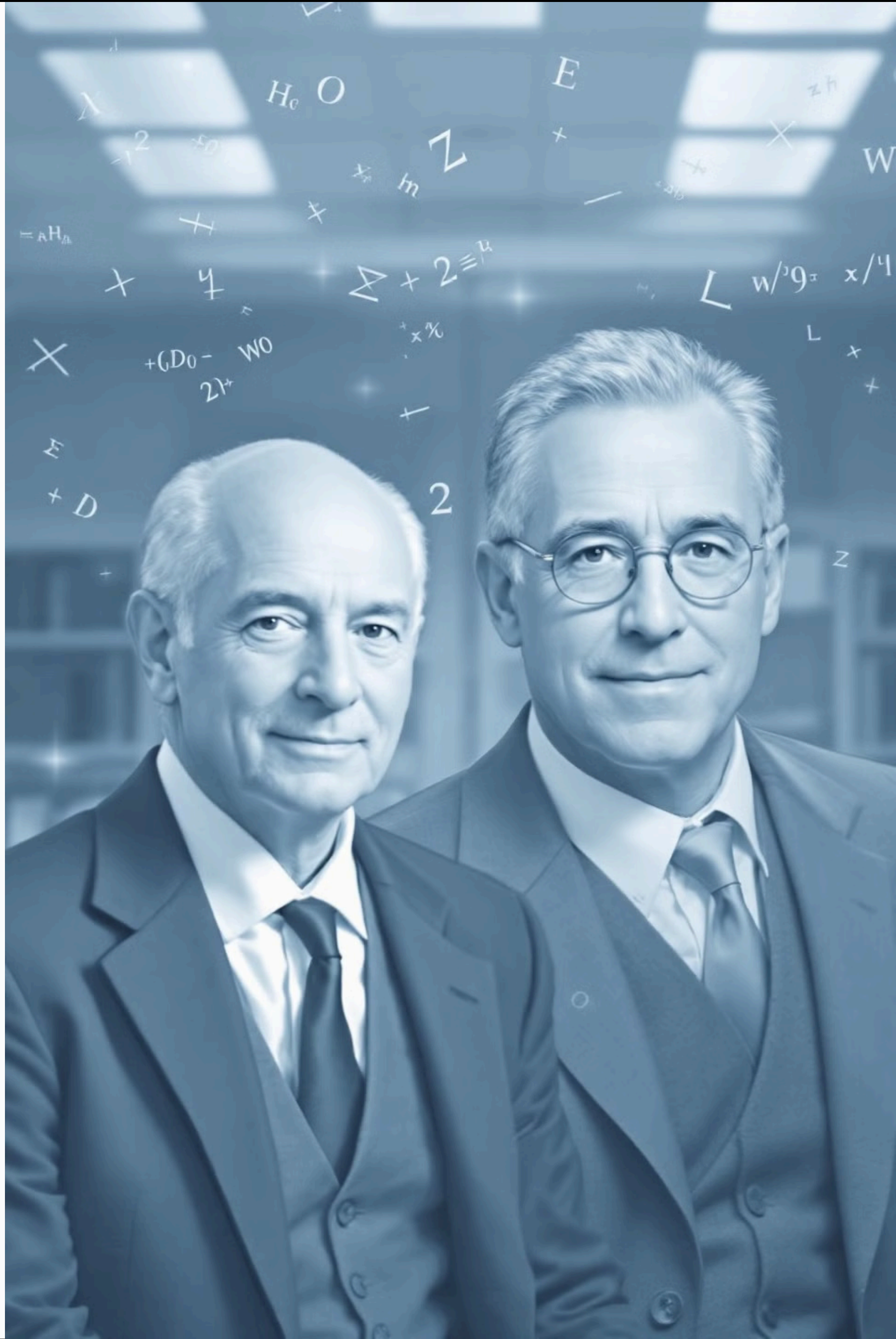
The isomorphism reveals that logical proofs correspond to programs, while logical propositions correspond to types.

2 Mathematical Foundation

Discovered independently by Haskell Curry and William Howard in the mid-20th century.

3 Practical Applications

This relationship forms the theoretical basis for modern type systems and proof assistants.



Core Idea

Logic ↔️ Programming

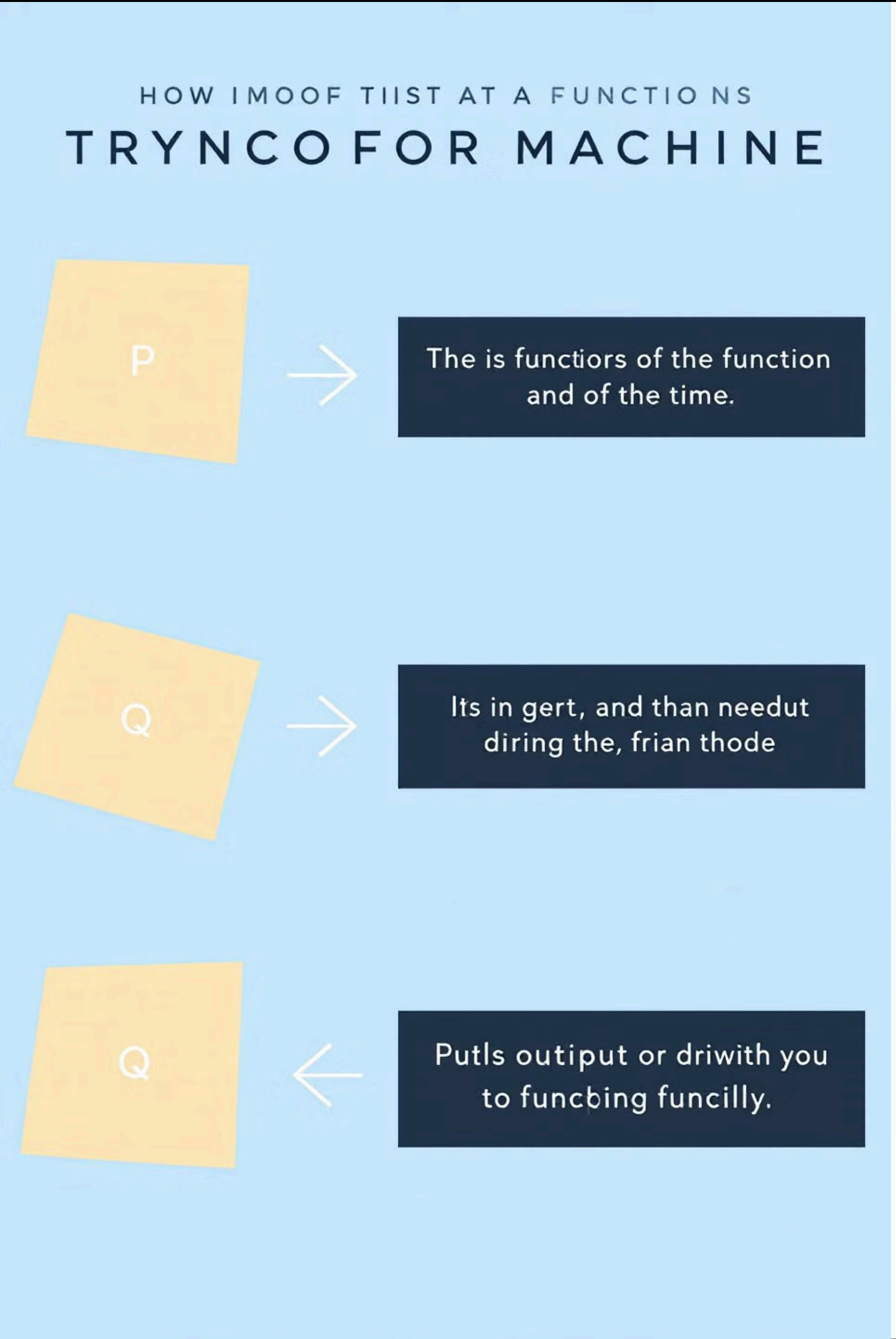
Mathematical logic and programming languages share a fundamental structure.

Propositions ↔️ Types

A logical proposition is equivalent to a type specification in programming.

Proofs ↔️ Programs

A logical proof corresponds to a program that satisfies a particular type.



Logical Implication (→) and Functions

1. **In Logic**

   P → Q means "If P is true, then Q is true"

2. **In Programming**

   A function type P -> Q transforms values of type P into values of type Q

3. **Example**

   Boolean negation function: not :: Bool -> Bool

Tuple Tugn

Sylgnes te 1gyion

Logical AND (∧) and Tuples

Logical Conjunction

P ∧ Q means "Both P and Q are true"

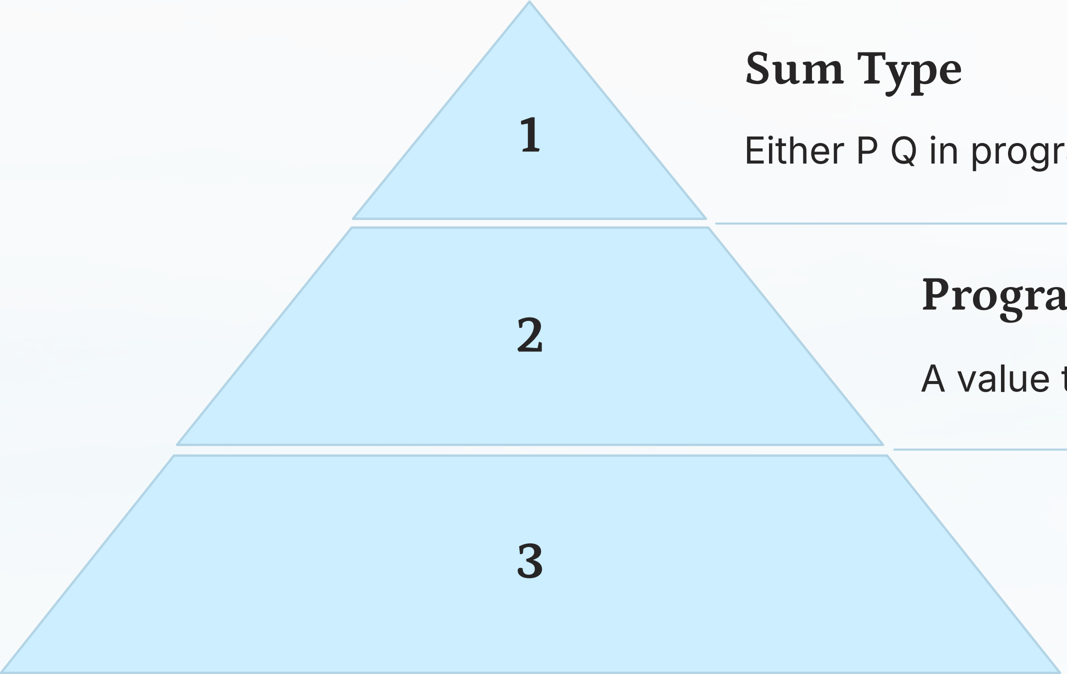
Programming Equivalent

A tuple (P, Q) contains values of both types P and Q

Implementation

pair :: P -> Q -> (P, Q) constructs a tuple from two values

Logical OR (∨) and Sum Types



Sum Type

Either P Q in programming languages like Haskell

Programming Concept

A value that can be either type P or type Q

Logical Meaning

$P \lor Q$ means "Either P is true or Q is true"

Sum types represent choice in programming, just as logical disjunction represents alternatives in logic.

Logical Falsehood (⊥) and the Empty Type

Logical Falsehood

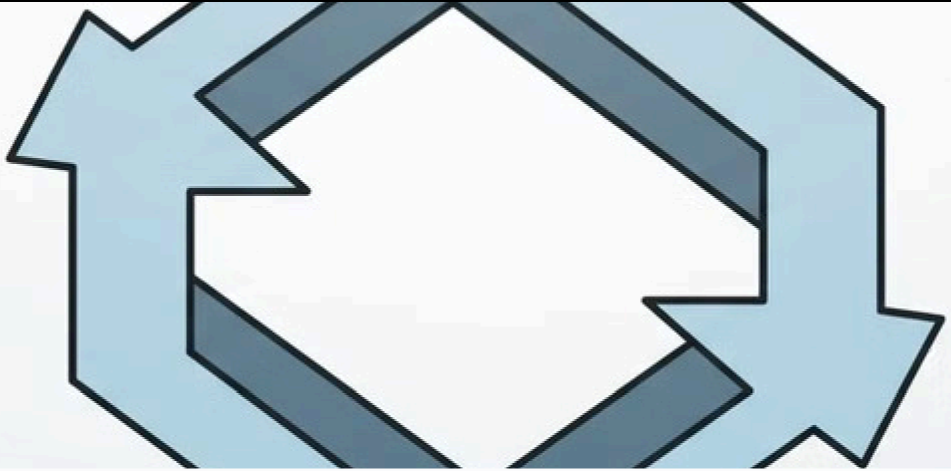
⊥ represents "False" - it can never be proven

Empty Type

The Void type has no values and cannot be instantiated

Implementation

In Haskell: data Void (a type with no constructors)



Logical Negation (¬P) and Functions to Void

Logical Negation

¬P means "P leads to a contradiction"

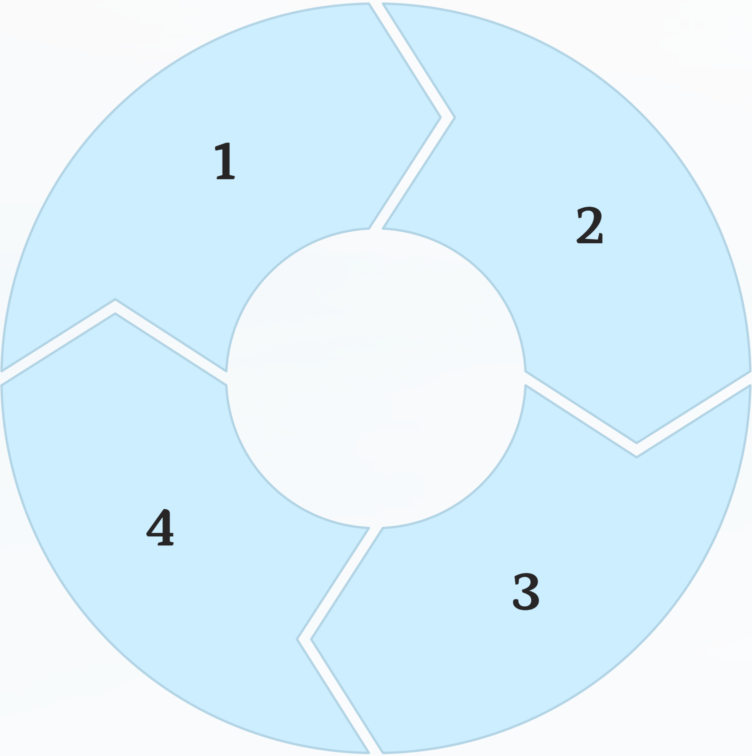
Programming Equivalent

P -> Void represents a function that can never return

Example Function

absurd :: Void -> a (cannot be called since Void has no values)

Polymorphism and Universal Quantification (∀)



Logical Universal

∀X. P(X) means "For all X, P(X) holds"

Polymorphic Functions

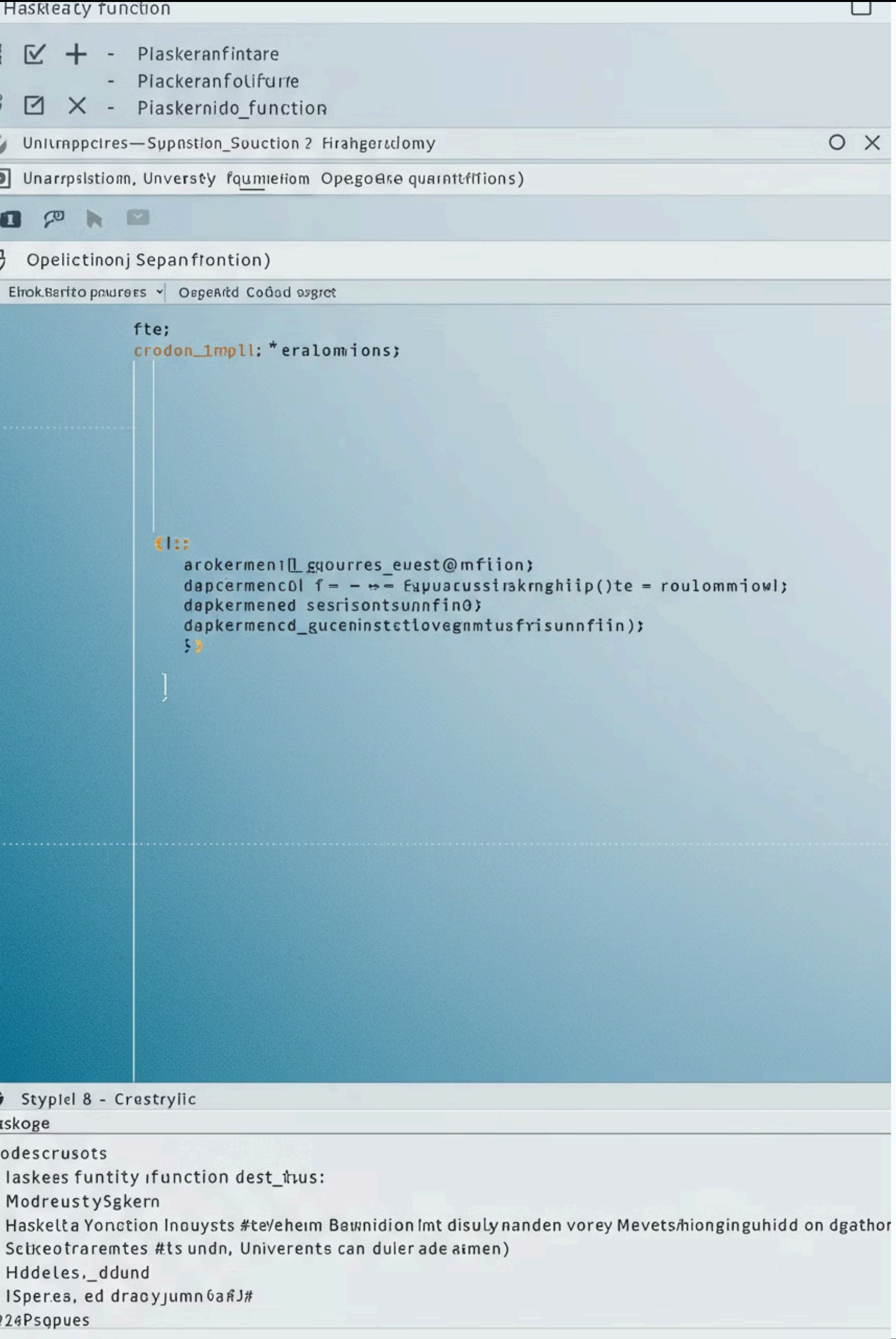
Functions that work with any type X

Programming Example

identity :: ∀X. X -> X works for all types

Type Variables

Represented by lowercase letters like 'a' in many languages



Example: Identity Function and Logical Universality

Logical Form	∀X. X → X
Meaning	For every type X, there exists a function from X to X
Haskell Type	id :: a -> a
Implementation	id x = x
Key Property	Works for any type, embodying universal quantification

Conclusion

1

Theoretical Foundation

Programs as proofs, types as propositions

2

Practical Applications

Powers proof assistants like Coq, Lean, and Agda

3

Broader Implications

Deepens our understanding of computation and logic

The Curry-Howard isomorphism bridges mathematical reasoning and computer programming, creating a foundation for verified software development.