Functional Programming

Yusuf M Motara, 2024 Edition

Administration

- 6 weeks
- 5 Practicals
    - Warm-up, Practical, and Homework sections
- 2 sets of tests
    - Best-of-2 for each set
- Daily problems, probabilistic assessment

Expectations

- Please be on time
- No distractions (cellphone, earbuds, etc)

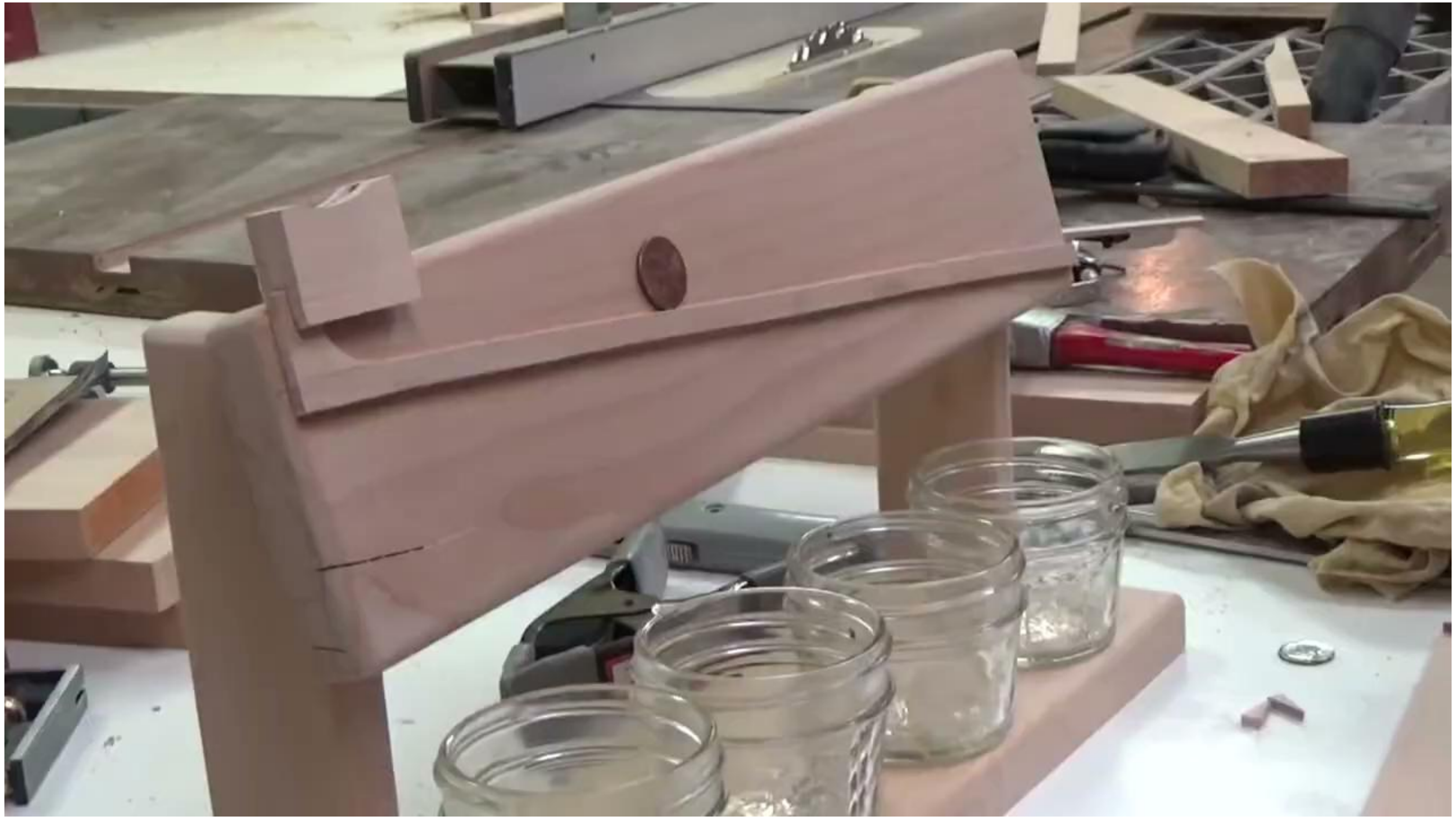
Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

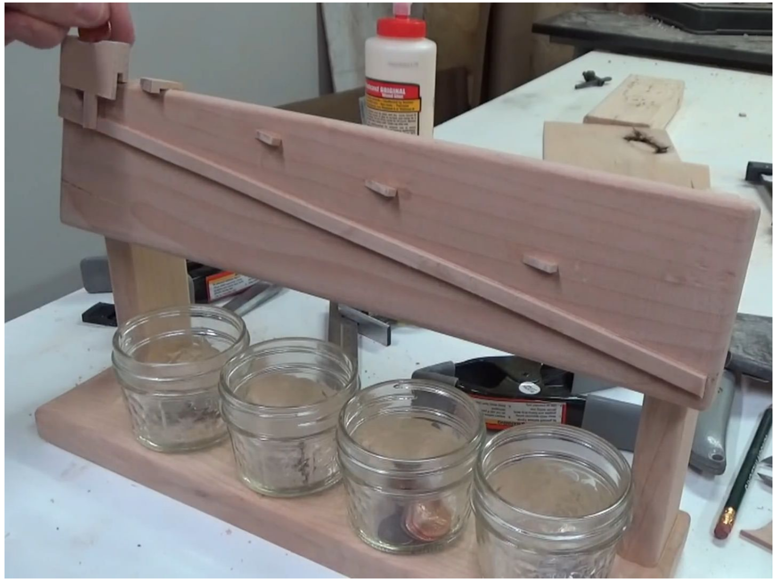
Why do we learn Functional Programming?

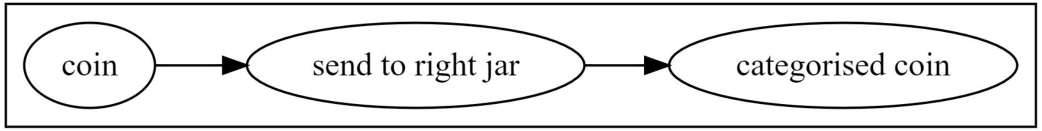
- Wave of the future
- Programs that cannot fail
- New tool in the toolbox

A gravity-based coin-sorting machine

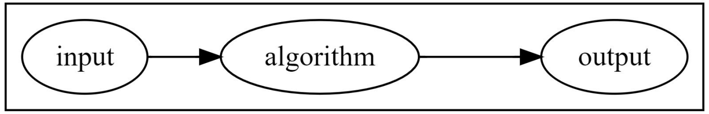


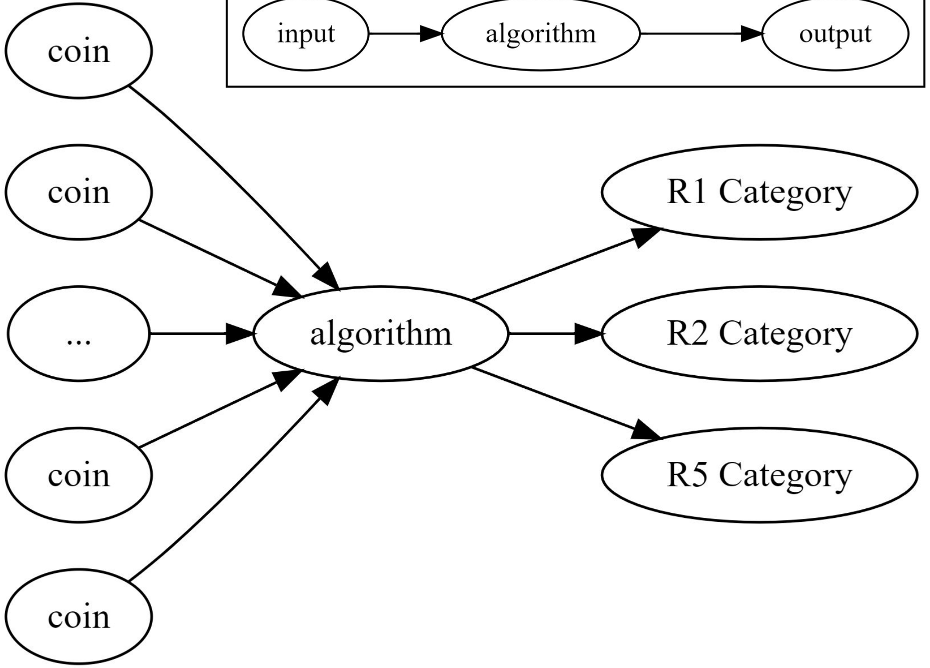
A gravity-based coin-sorting machine

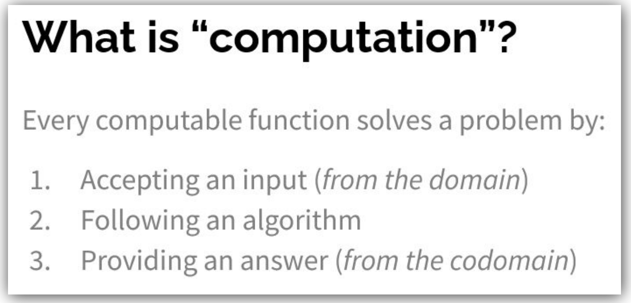




A gravity-based coin-sorting machine







What is "computation"?

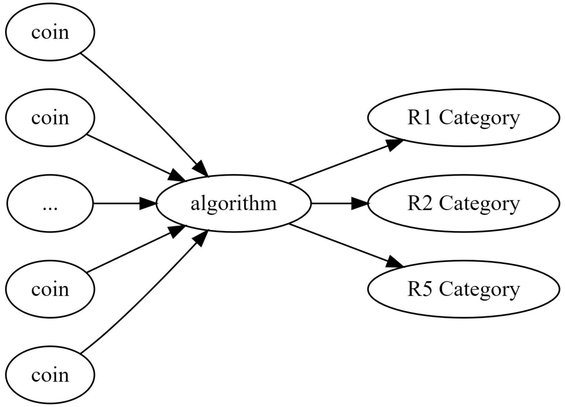
Every computable function solves a problem by:

1. Accepting an input (*from the domain*)
2. Following an algorithm
3. Providing an answer (*from the codomain*)

A gravity-based coin-sorting machine

$$\begin{tikzcd}\text{coin} \arrow[r] & \text{send to right jar} \arrow[r] & \text{categorised coin}\end{tikzcd}$$

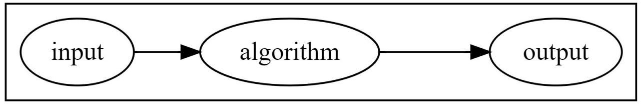
$=$



$$\begin{tikzcd}\text{input} \arrow[r] & \text{algorithm} \arrow[r] & \text{output}\end{tikzcd}$$

What is “functional programming”?

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques



A way of using only **functions** and **symbols** to create programs

“Functions” in functional programming are **pure computational** things, e.g:

$$x \mapsto 2x$$

They are as reliable as **basic addition**!

What is a “pure computational” function?

Features:

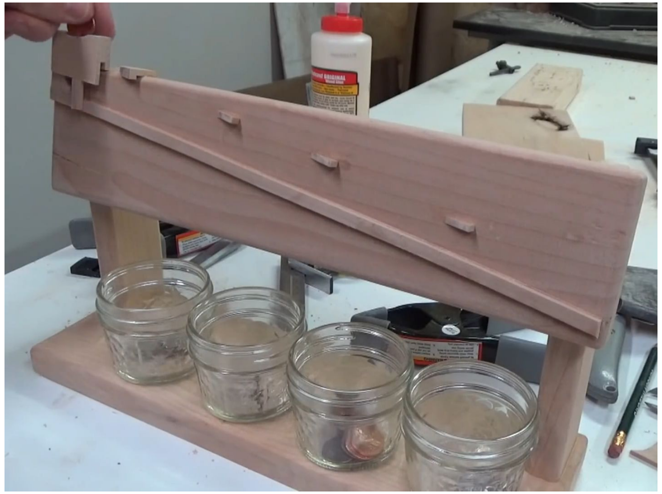
- Maps a domain to a codomain
  - We use “arrow syntax” to show domain and codomain
  - Tells us *what kinds of things* the input and output are
- Either terminates (i.e. provides an answer), or does not terminate (i.e. loops infinitely, or throws an exception)
  - Do you remember this from *Theory of Computation*?
  - If it *always* terminates properly, it is a **total function**. Otherwise, it is a **partial function**.
- Maps each element of the domain to an element of the codomain—*and <u>does nothing else</u>* (i.e. is “pure”)*!*
  - We use “barred arrow syntax” to show exactly how this is done
  - Tells us *what computation* generates the output

$$\mathbb{N} \to \mathbb{R}$$

$$x \mapsto x^2 + \frac{x}{3} + 1$$

What is a “pure computational” function?

- Maps a domain to a codomain
  - We use “arrow syntax” to show domain and codomain
  - What is the domain? What is the codomain?
- Either terminates (i.e. provides an answer), or does not terminate (i.e. loops infinitely, or throws an exception)
  - If it *always* terminates properly, it is a **total function**. Otherwise, it is a **partial function**.
  - Is this *total*, or *partial*?
- Maps each element of the domain to an element of the codomain
  - Can you see how this is done?
  - Is this “function” pure?



Different core abstractions in imperative programming!!

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Functions are the primary way that we manipulate data.

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Binding is about how symbols are linked to values.

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Binding is about how symbols are linked to values.

$$\text{Let } x = 5 \text{ in } x \mapsto x^2 + \frac{x}{3} + 1$$

Where is the binding happening?

What can happen after a binding?

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Binding is about how symbols are linked to values.

$$\text{Let } x = 5 \text{ in } x \mapsto x^2 + \frac{x}{3} + 1$$

In a **pure computational** function, when would binding be done?

Where would binding be done?

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Algebraic data types help us to organize our data into custom abstractions that make sense.

The most interesting custom abstractions that you have been introduced to, up to now, have been “objects”. You have also been told about “structs”, “tuples”, and other ways of *grouping together related data*.

All of these have been fundamentally the same.

We will introduce you to a way of making abstractions that, for the first time, are actually real data types and not “blobs of related stuff”.

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Lastly, we will cover **function techniques**, **design**, and **functional programming practice**.

Function techniques give us ways to combine, sequence, and otherwise manipulate functions. This allows us to manipulate data in more readable and maintainable ways.

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Here is a function to calculate the VAT on shopping: $x \mapsto 1.15x$

And here is a function to add on the cost of a plastic bag: $y \mapsto y + 0.28$

If you want to calculate the total price that you pay at the till, you can write a new function… but function techniques let you just combine *existing* functions instead, saving you time and effort 🎉.

Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Functional Programming Practice	Function Techniques

Design is about the ways in which we can design functional programs. Since we use different core abstractions, we can't design them in the same way that we design imperative programs.

What will be **really** important?

1. Practice
2. Practice
3. Practice
4. Practice
5. Practice

What will be **really** important?

1. Practice
2. Practice
3. Practice
4. Practice
5. Practice

Faith

Questions?

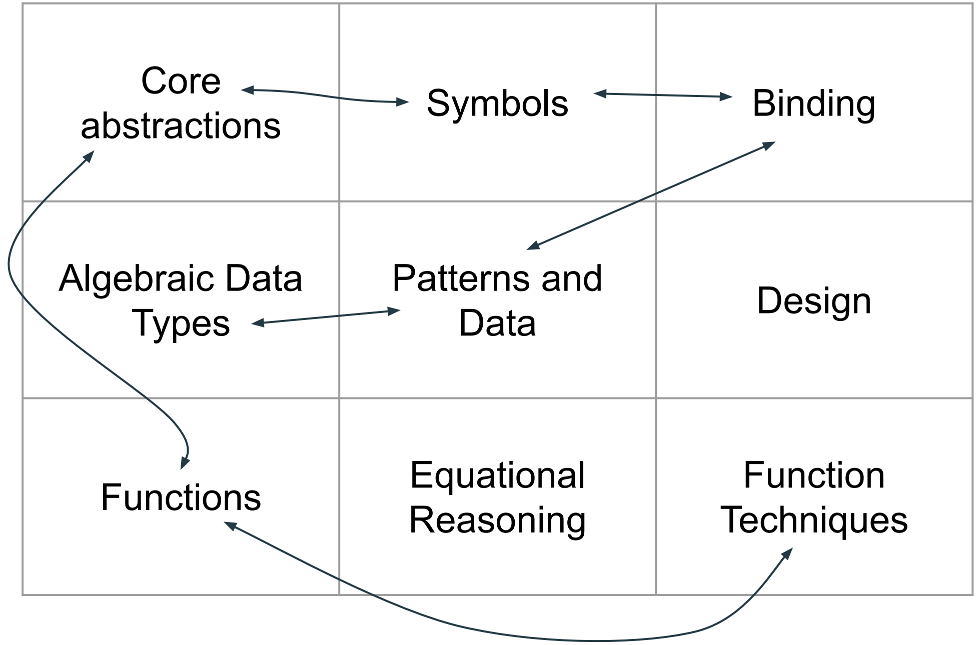
Overview

Core abstractions	Symbols	Binding
Algebraic Data Types	Patterns and Data	Design
Functions	Equational Reasoning	Function Techniques

Equational reasoning is a powerful method, only usable in functional programs, that we can use to "debug" and reason about functional programming.

Using equational reasoning, we can *prove* correctness of our programs—and in fact, it is so powerful that some functional languages can **guarantee** that any program which compiles, *will never crash*! That kind of strong guarantee is impossible to make for any but the most trivial imperative programs.

Overview



Questions?