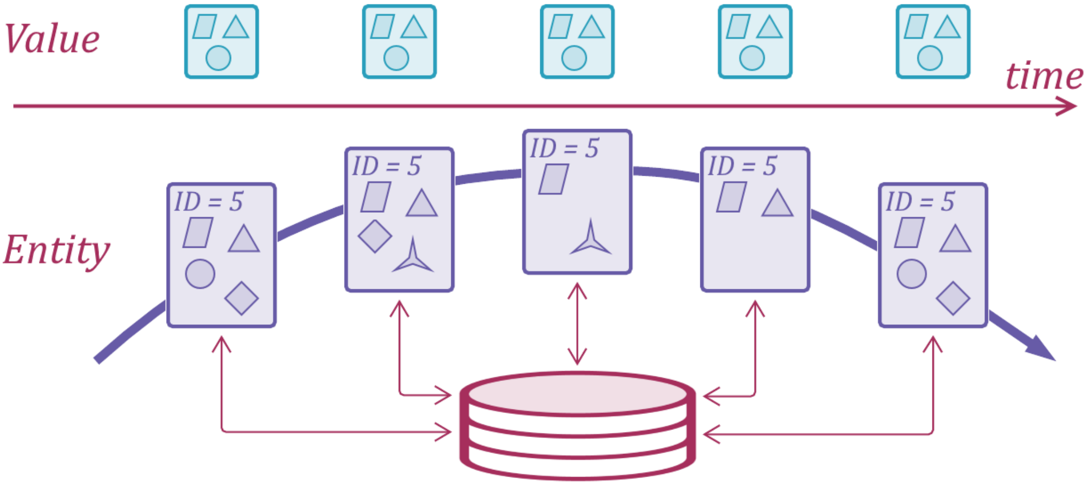
Treating All Objects as Values

Zoran Horvat

CEO AT CODING HELMET

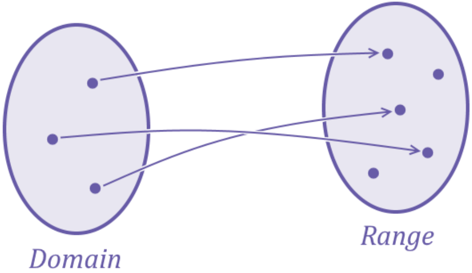
@zoranh75 http://csharpmentor.com

Values vs. Entities



Function as a Mapping

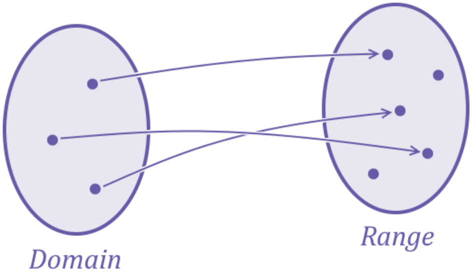
Mathematics



Programming

Function as a Mapping

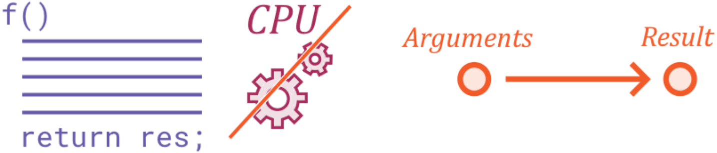
Mathematics



Domain

Range

Programming



`f()`

`return res;`

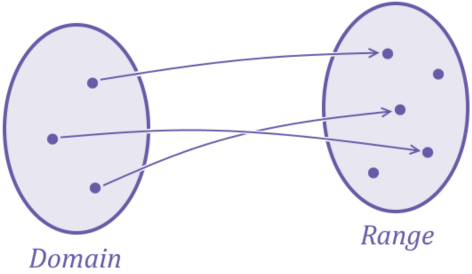
CPU

Arguments

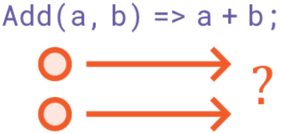
Result

Function as a Mapping

Mathematics

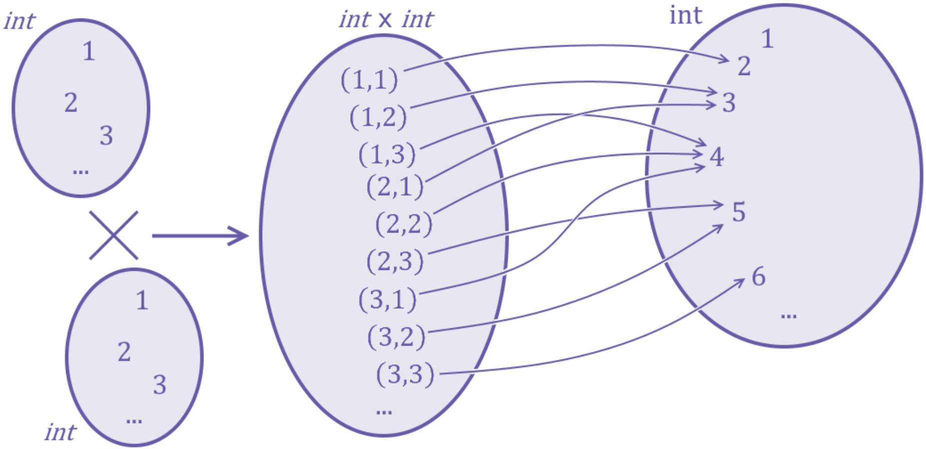


Programming



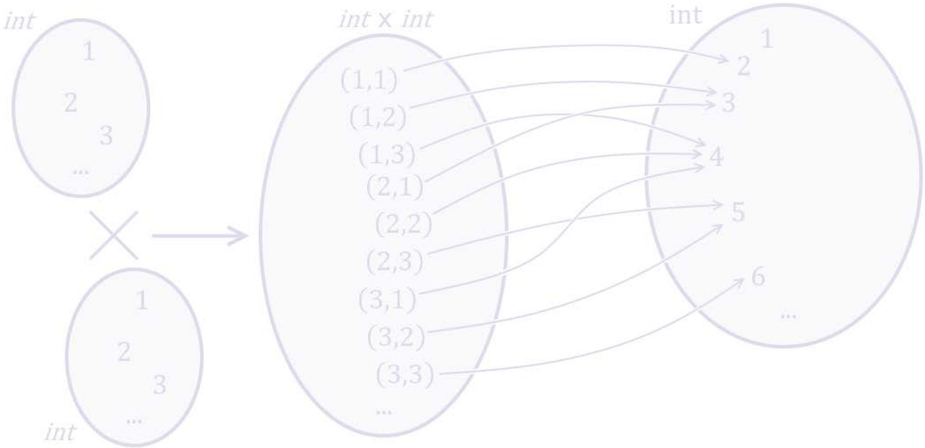
```
Add(a, b) => a + b;
```

Mapping Multiple Arguments



Mapping Multiple Arguments

```
int Add( (int a, int b) tuple) => tuple.a + tuple.b;
```



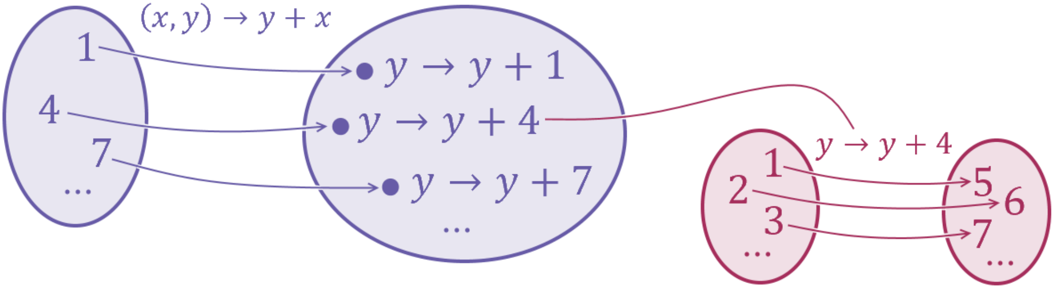
Understanding Currying

Haskell Curry

Currying
Transforming a multi-argument function into a series of one-argument functions

Haskell programming language
Named after Haskell Curry

Understanding Currying



Currying in F#

```
let add a b = a + b
int → (int → int)
```

Currying in C#

No built-in support

Inventing Pure Functions

Receives and returns values

Programmatic functions correspond to mappings in mathematics

Values are immutable

Values can be compared for equality

Inventing Pure Functions

Receives and returns values

No observable side effects

Function produces no side effects meaningful to the program

Function only depends on its arguments

Inventing Pure Functions

Receives and returns values

No observable side effects

Always returns the same result

Function returns the same value when invoked with same arguments again

Inventing Pure Functions

Receives and returns values

No observable side effects

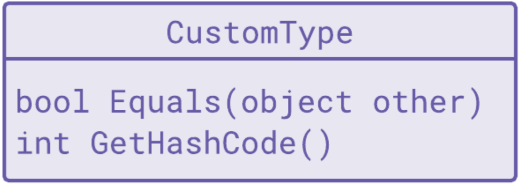
Always returns the same result

Referentially transparent

Pure function can be replaced with the value it produces for given arguments

Only a side-effect-free function operating on values can be pure

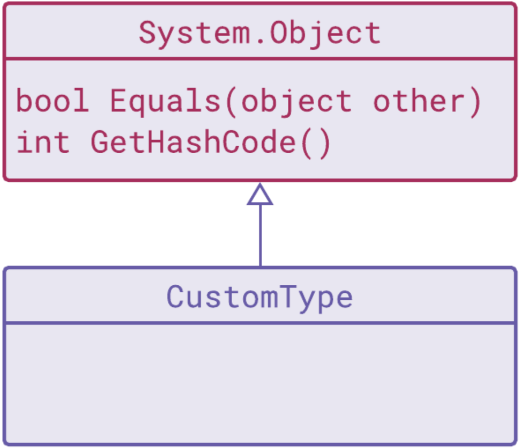
Value Equivalence in .NET



CustomType

```
bool Equals(object other)
int GetHashCode()
```

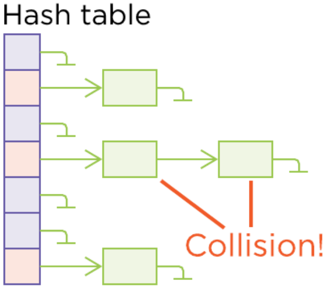
Value Equivalence in .NET

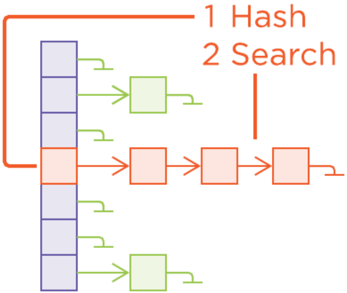


System.Object

bool Equals(object other)
int GetHashCode()

CustomType


Hash table
Collision!


1 Hash
2 Search

Equivalence relation

binary

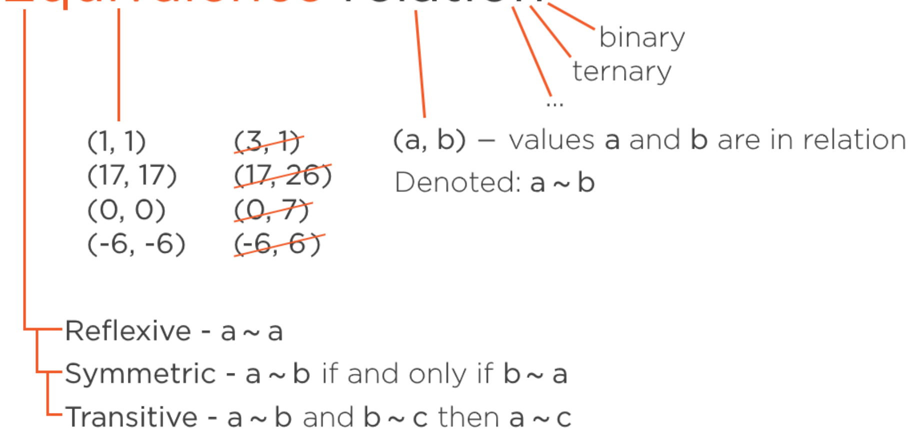
ternary

...

(a, b) – values **a** and **b** are in relation

Denoted: **a ~ b**

Equivalence relation



binary

ternary

...

(1, 1) ~~(3, 1)~~

(17, 17) ~~(17, 26)~~

(0, 0) ~~(0, 7)~~

(-6, -6) ~~(-6, 6)~~

(a, b) – values a and b are in relation

Denoted: $a \sim b$

Reflexive - $a \sim a$

Symmetric - $a \sim b$ if and only if $b \sim a$

Transitive - $a \sim b$ and $b \sim c$ then $a \sim c$

Equivalence relation

Equality (programming)

Is an object **a** equal to another object **b**?

`expr` equal to **5**

a equal to **3**

5 not equal to `expr`

3 not equal to **a**

- Reflexive - a ~ a
- Symmetric - a ~ b if and only if b ~ a
- Transitive - a ~ b and b ~ c then a ~ c

Equivalence relation

Equality (programming)

`Object.Equals()` - returns `True` on equal objects

`Object.GetHashCode()` - returns same value from equal objects

Used to define equivalence relation on a single class

- Reflexive - a ~ a
- Symmetric - a ~ b if and only if b ~ a
- Transitive - a ~ b and b ~ c then a ~ c

Summary

`GetHashCode` **and** `Equals` **methods**

- Implement equivalence relation
- Lets you use an object as the key

Value-typed semantic

- Class implements equivalence via `GetHashCode` and `Equals`
- Class is immutable

Summary

Implementing pure functions

- Arguments are value objects
- Return value is a value object
- No observable side effects

Referential transparency

- Applied to pure functions
- Function is interchangeable with the value it produces
- No need to call the function twice with same argument values

Next module:

Controlling Execution Flow with Pattern Matching