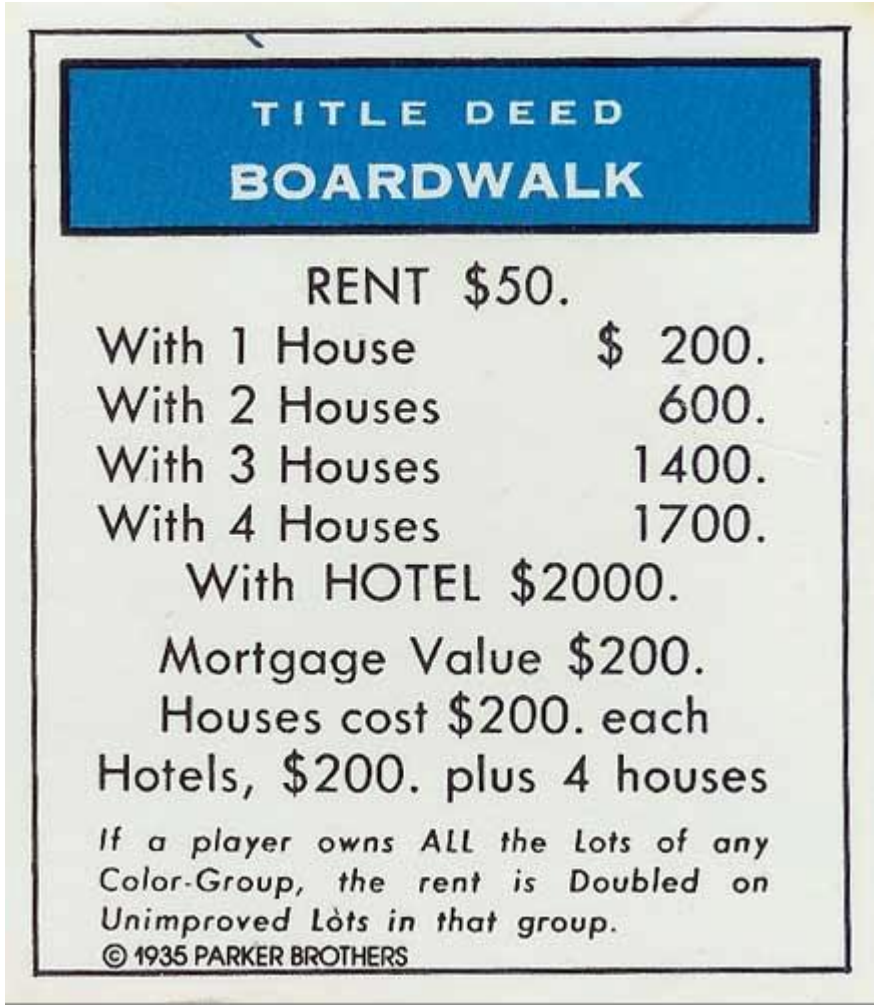
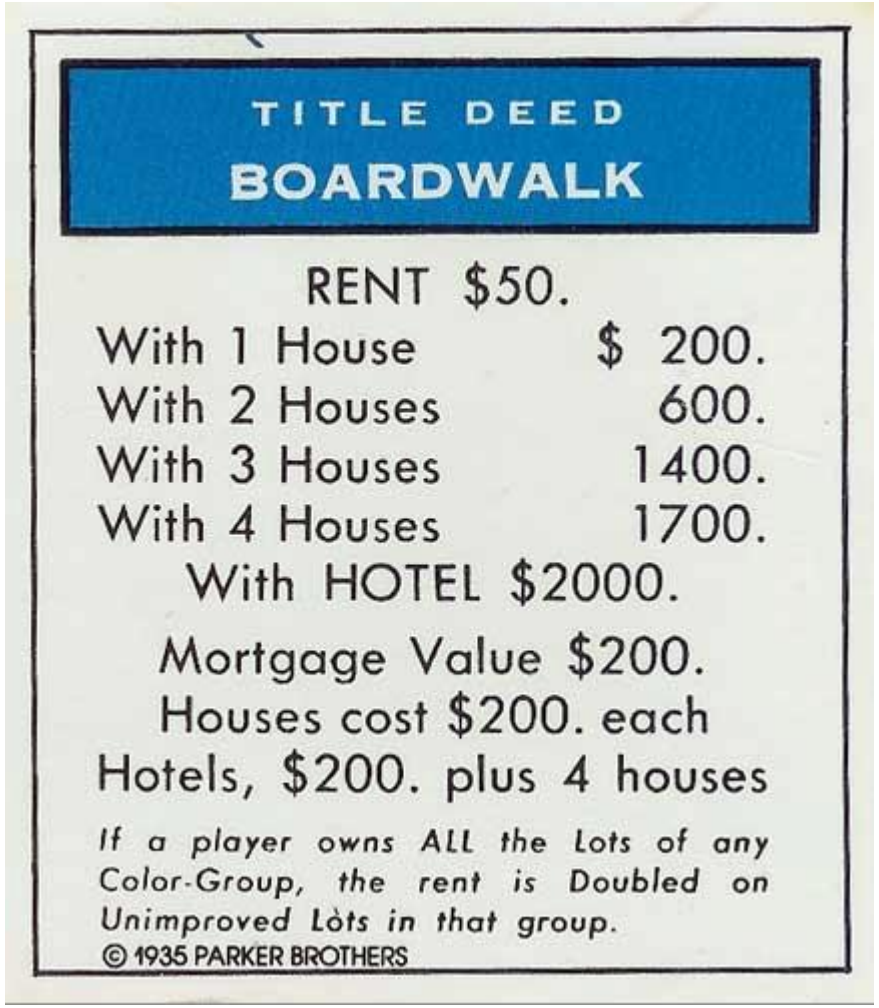
Thanks to our Sponsors!

To connect to wireless

1. Choose Uguest in the wireless list
2. Open a browser. This will open a Uof U website
3. Choose Login



Price
Rent
HouseCost
MortgageValue
IsMortgaged
CanMortgage
Houses
HasHotel
CanBuildHouse
Owner



Data, State, Derived Value

Price

Rent

HouseCost

MortgageValue

IsMortgaged

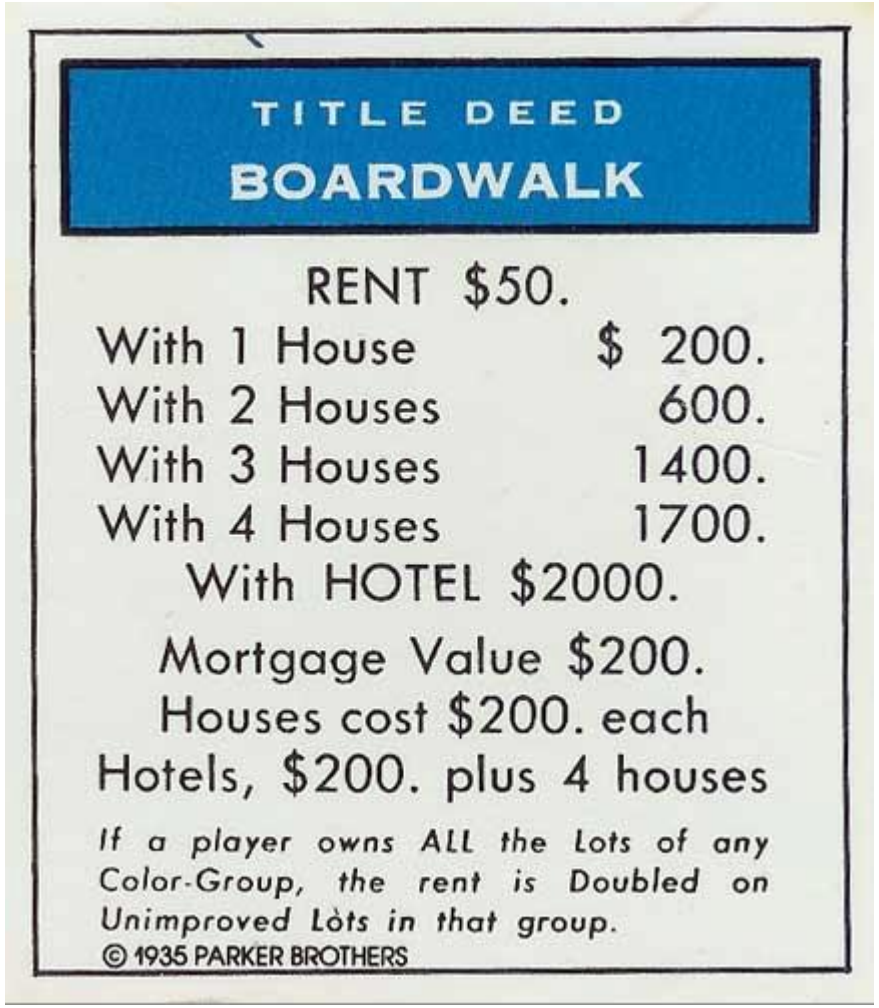
CanMortgage

Houses

HasHotel

CanBuildHouse

Owner



Rent

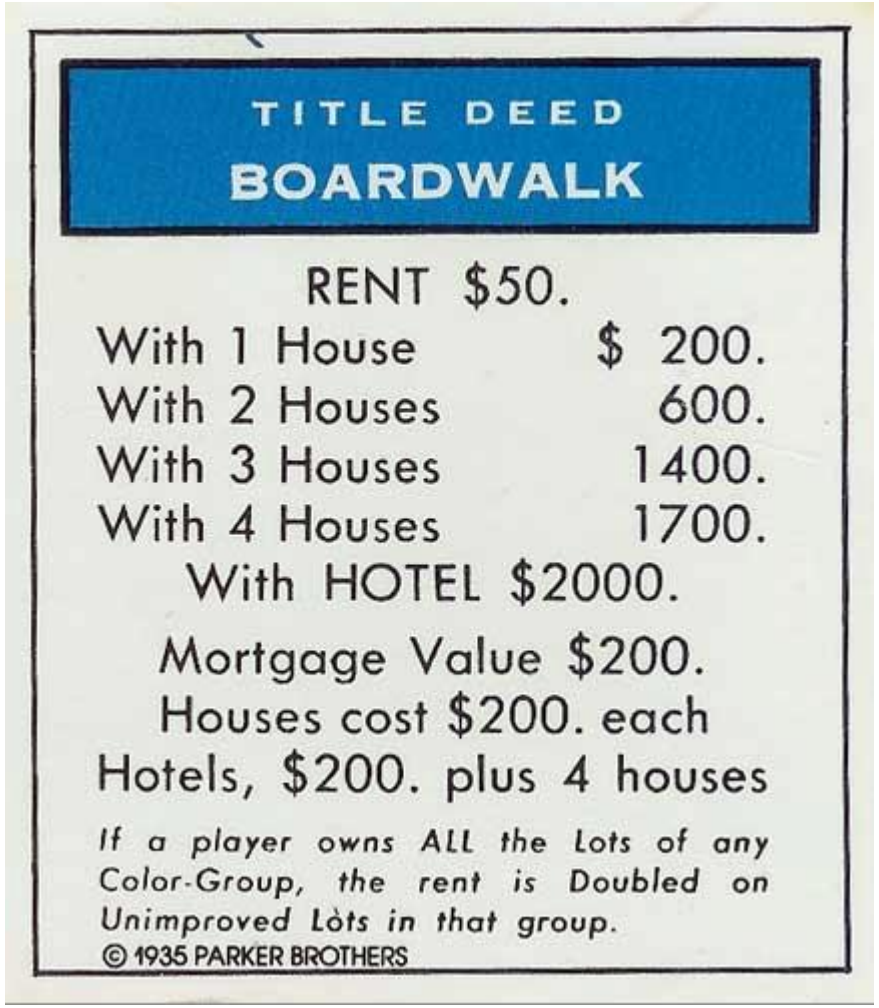
- Rent doubles if you own all values in the same color-group

CanMortgage

- “Before an improved property can be mortgaged, all the buildings on all the properties of its color-group must be sold back to the Bank.”

CanBuildHouse

- Must own all properties of color group
- None of the properties in the color-group can be mortgaged
- Player must have enough money
- Must have enough houses in the bank



Rent

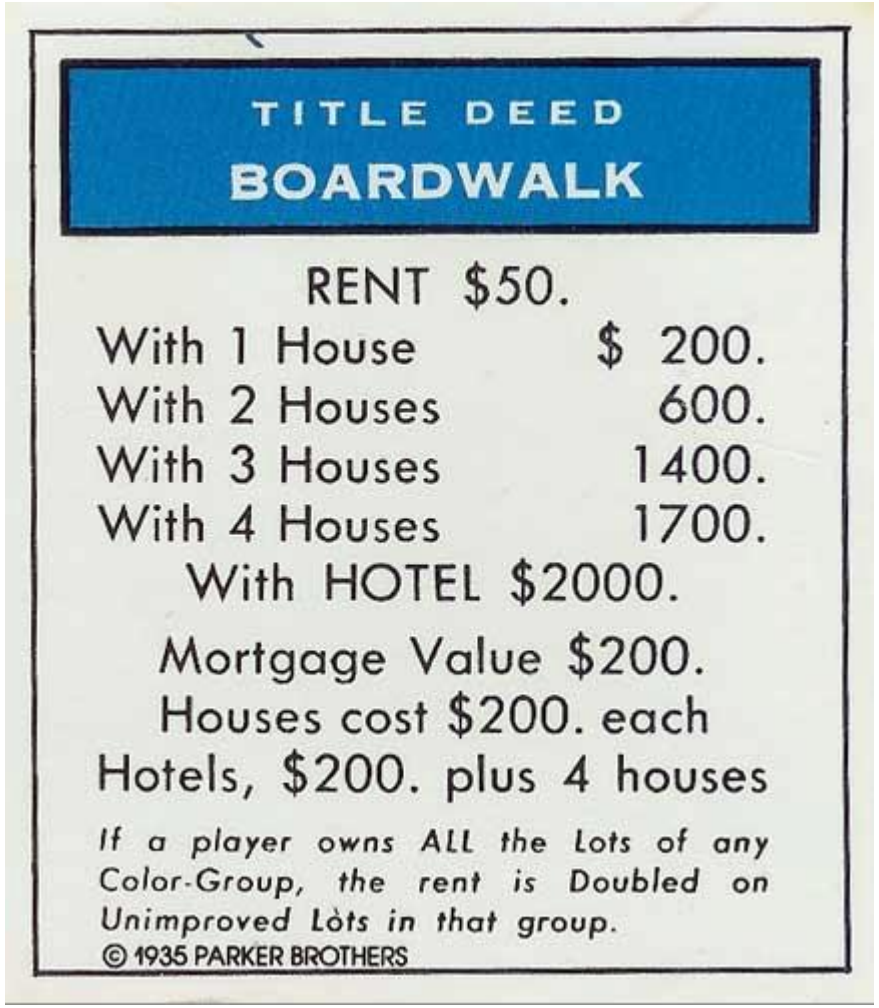
- Rent doubles if you own all values in the same color-group

CanMortgage

- "Before an improved property can be mortgaged, all the buildings on all the properties of its color-group must be sold back to the Bank."

CanBuildHouse

- Must own all properties of color group
- None of the properties in the color-group can be mortgaged
- Player must have enough money
- Must have enough houses in the bank



Data, State, Derived Value

Price

~~Rent~~

HouseCost

MortgageValue

IsMortgaged

~~CanMortgage~~

Houses

HasHotel

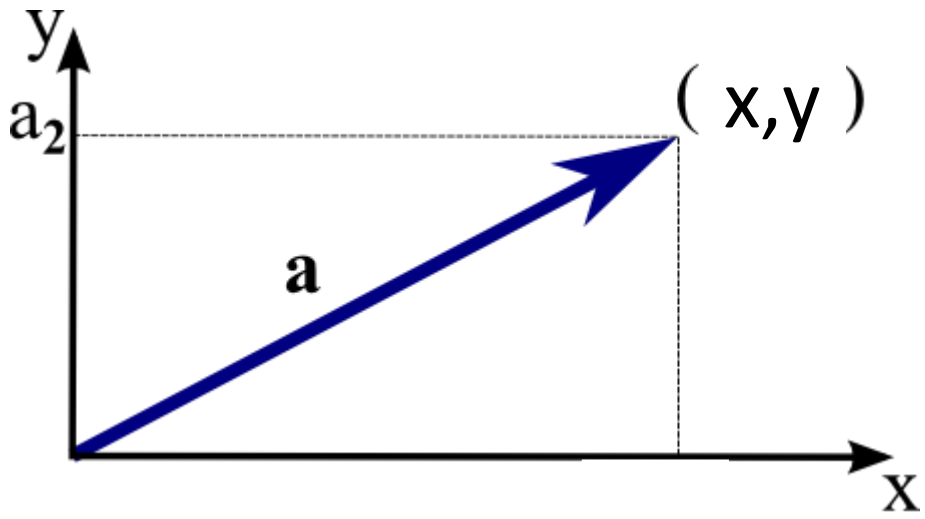
~~CanBuildHouse~~

Owner

The problem with object-oriented languages is they’ve got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle.

--Joe Armstrong, interviewed in Coders at Work

Data, State, Derived Value, Methods



X

Y

Negative

Normalized

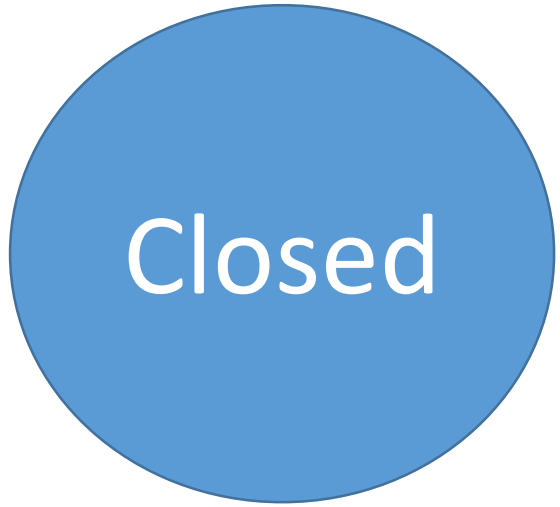
Length

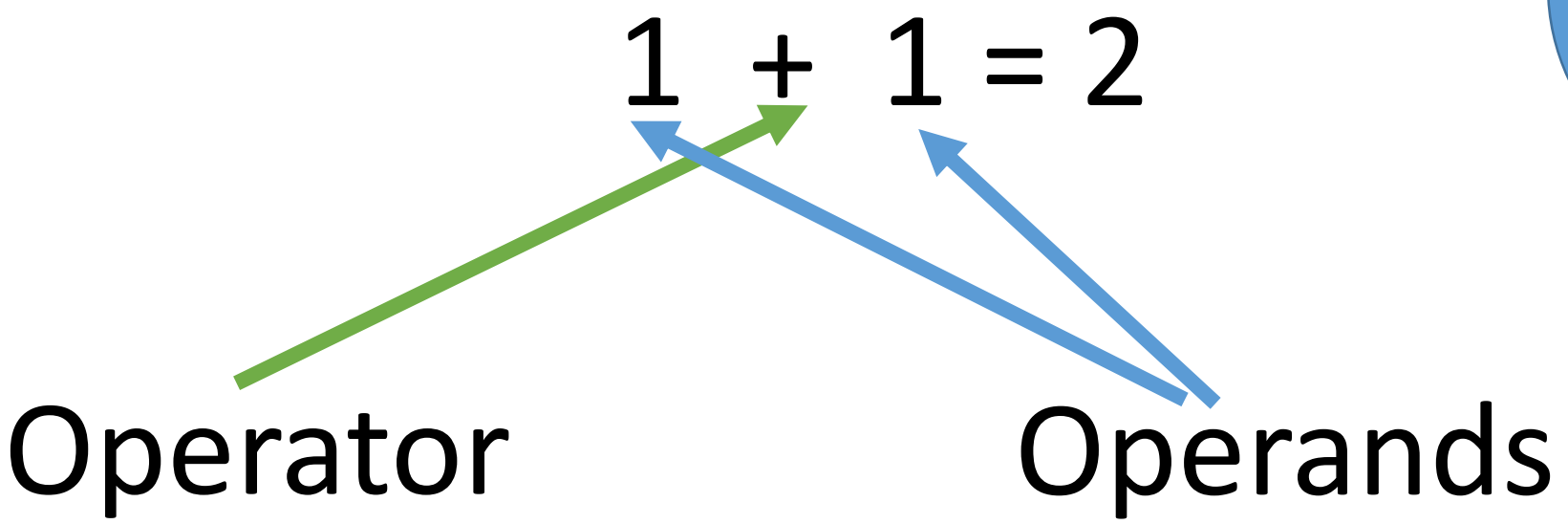
float Dot(Vector2D)

Vector2D Add(Vector2D)

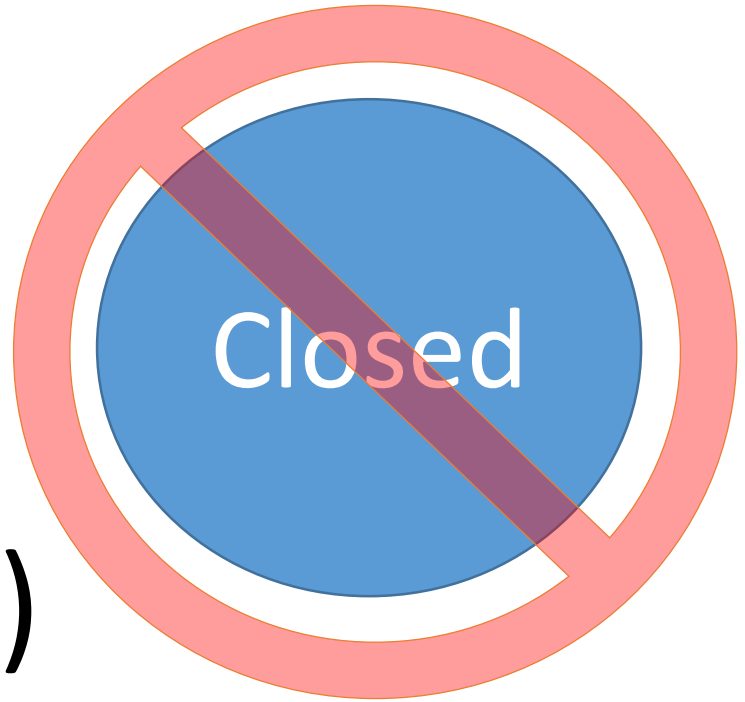
Vector2D Cross(Vector2D)

Closed operations



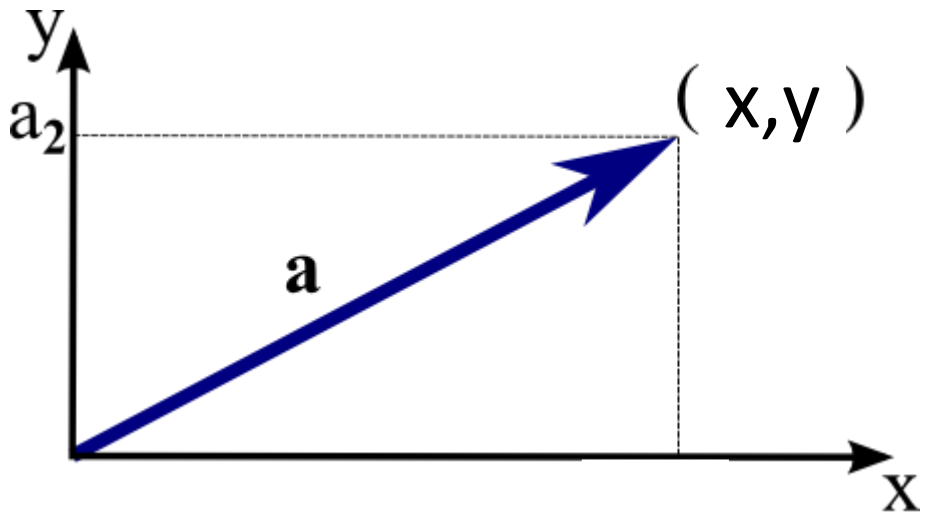


Closed operations



5 / 4 = .8 (not an int)

Data, State, Derived Value, Methods



X

Y

Negative

Normalized

Length

float Dot(Vector2D)

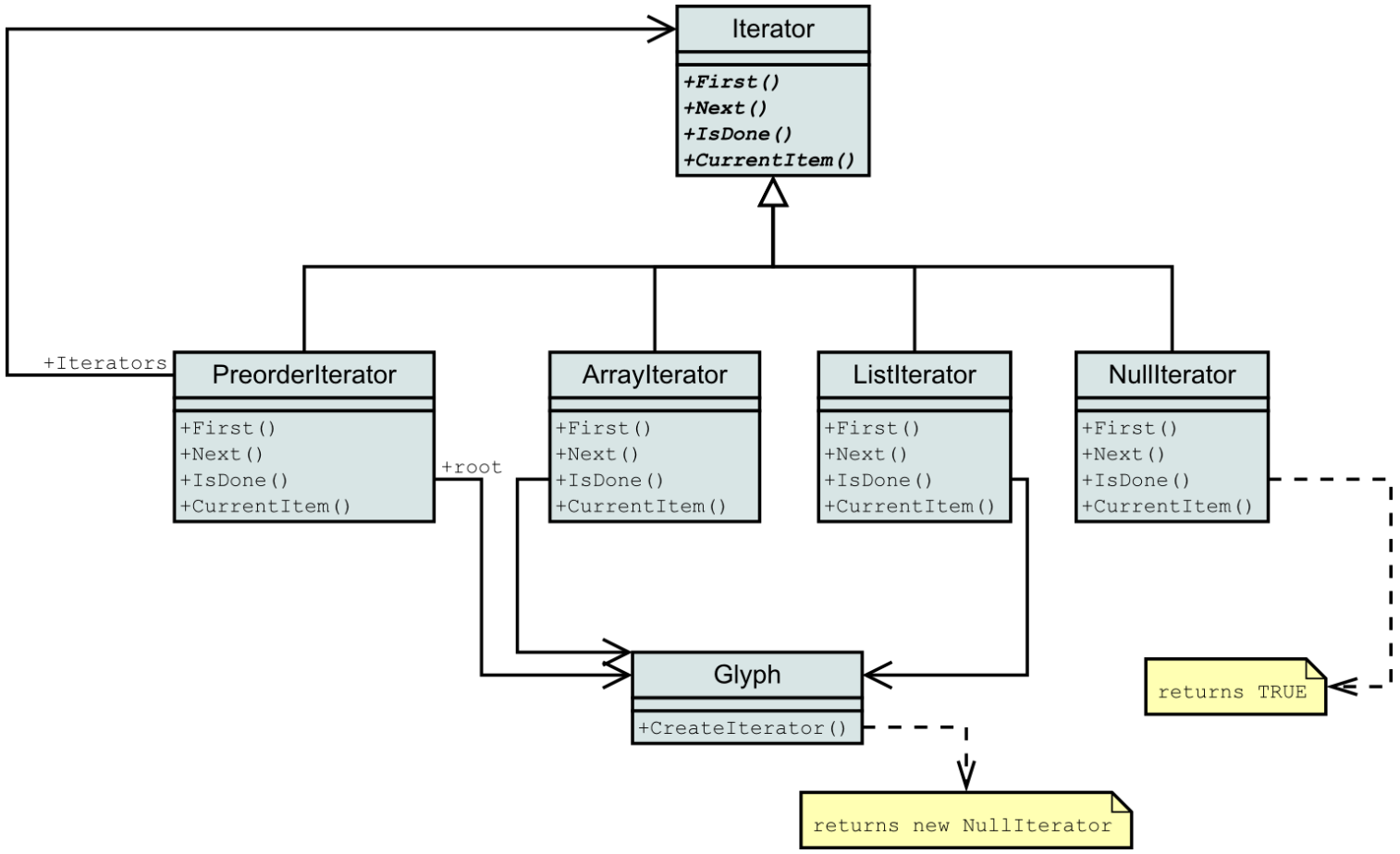
Vector2D Add(Vector2D)

Vector2D Cross(Vector2D)

Local concerns make great objects. Global concerns make lousy objects.

Interactions define software far more than the individual elements. OOP focuses on the elements, not the interactions.

UML Class Diagrams



How we actually design software

- Elements
- Responsibilities
- Interactions/Dependencies

PKG

Webdeploy

1) Input: Url, Label,
2) Creates Package
3) Upload Package

INT

- manifest
- Resources*
- Can update Package?

Assets Package

ACC

PPE

Label, Web.config Path,

1) Pull Package
2) Send to origin servers
3) Modify Web.config

Prod

1) Stack ID
2) Env
3) Go Promote Label [1 | CNT. (Rev)]

↑
In environment Label

CNS_CVS

/CNS_CVS/ BuildID/ _____

CNS_CVS_Assets

- Back button
- Bookmarks / Deep links
- Google Analytics
- Disqus
- Crawl - Pagination

- Search Pages

isInSchedule(storeId, time, ...)

Lead time

Pick Up Times

Meal	Baker
- Is any - Blackouts - Holidays	store hours by hour

Order

Order Item

Oven Capacity

- Limits on # of items in a timeslot (for some items)

hasCapacity(store, type, qty, with)

Capacities on date

Budget

- Standards (Generic)
- $, Quantity, Rate

Commitment

Change / Editing, Pending, Approved

→ Commitment

Expectation
Commitment
Change Order

Tracking

- Resource
- Allocated to budget line
- Unallocated

Reconciliation

- Quantity
- Rate
- Un Planned
- Change Order

Invoice

Budge
Prj Ava
Assignm
Timeshe
Report
Invoic

1) Design Budget to work w/ commitments

- Explanation Needs to be at line level

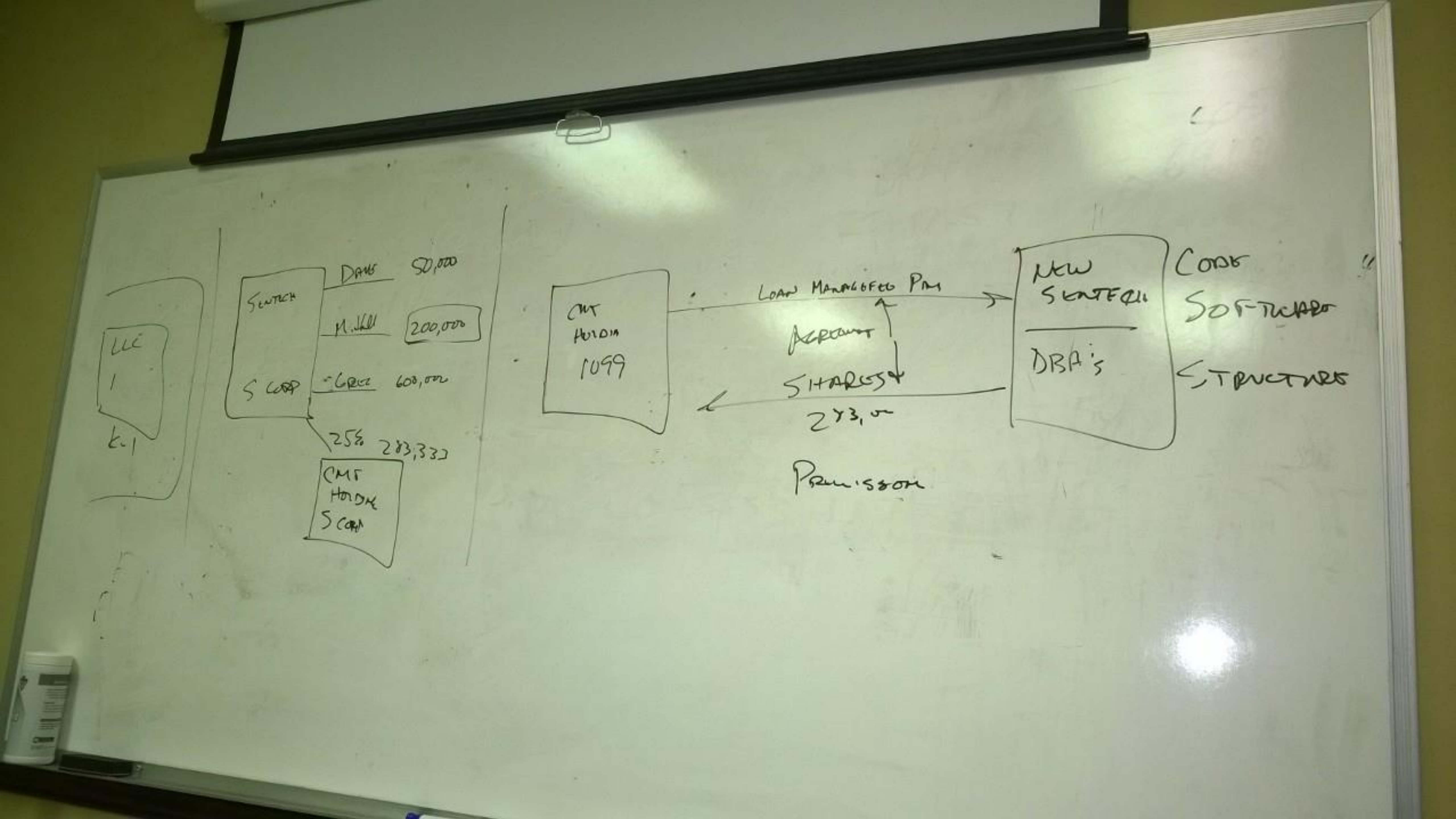
PrjTask

Budget

*CO

*Line Item
-Std
-Qty
-Rate

Engineer	2			
Tech	20			
Truck	1			


LLC
K-1
S CORP
DAVE 50,000
200,000
600,000
25%
283,333
CMT HOLDING
S CORP
CMT HOLDING
1099
SHARES
DBA's
CODE
SOFTWARE
STRUCTURE
PERMISSION

How we actually design software

- Elements
- Responsibilities
- Interactions/Dependencies

So what are my other tools?

- Object Oriented Programming
- Procedural Programming
- Functional Programming
- Sets/Relational Programming
- Data driven
- Evented architectures

Some examples

- Partial Function Application
- Pattern Matching
- Cloning
- List Monad (LINQ or Lodash)
- Reactive Programming
- Sets

What about services? OOP does those too.

```
public class NarratorService {

  private readonly ITopicExtractor extractor;
  private readonly IOutliner outliner;

  public NarratorService(ITopicExtractor extractor, IOutliner outliner) {
    this.extractor = extractor;
    this.outliner = outliner;
  }

  public string Narrate(AncestryDataModel data) {
    //Some stuff happens that uses the extractor and the outliner
  }
}
```

What we wrote

```
public class NarratorService {

  private readonly ITopicExtractor extractor;
  private readonly IOutliner outliner;

  public NarratorService(ITopicExtractor extractor, IOutliner outliner) {
    this.extractor = extractor;
    this.outliner = outliner;
  }

  public string Narrate(AncestryDataModel data) {
    //Some stuff happens that uses the extractor and the outliner
  }
}
```

What the function needs

```
string Narrate(ITopicExtractor extractor, IOutliner outliner, AncestryDataModel data) {
  //Some stuff happens that uses the extractor and the outliner
}
```

Partial Function Application

In Javascript

```
function narrate(topicExtractor, outliner) {
  return function(data) {
    //Some stuff happens that uses the extractor and the outliner
  };
}
```

In Scala

```
def narrate(extractor: ITopicExtractor, outliner IOutliner)(data: AncerstryDataModel): String =
  //Some stuff happens that uses the extractor and outliner
```

```
public class NarratorService {

  private readonly ITopicExtractor extractor;
  private readonly IOutliner outliner;

  public NarratorService(ITopicExtractor extractor, IOutliner outliner) {
    this.extractor = extractor;
    this.outliner = outliner;
  }

  public string Narrate(AncestryDataModel data) {
    //Some stuff happens that uses the extractor and the outliner
  }
}
```

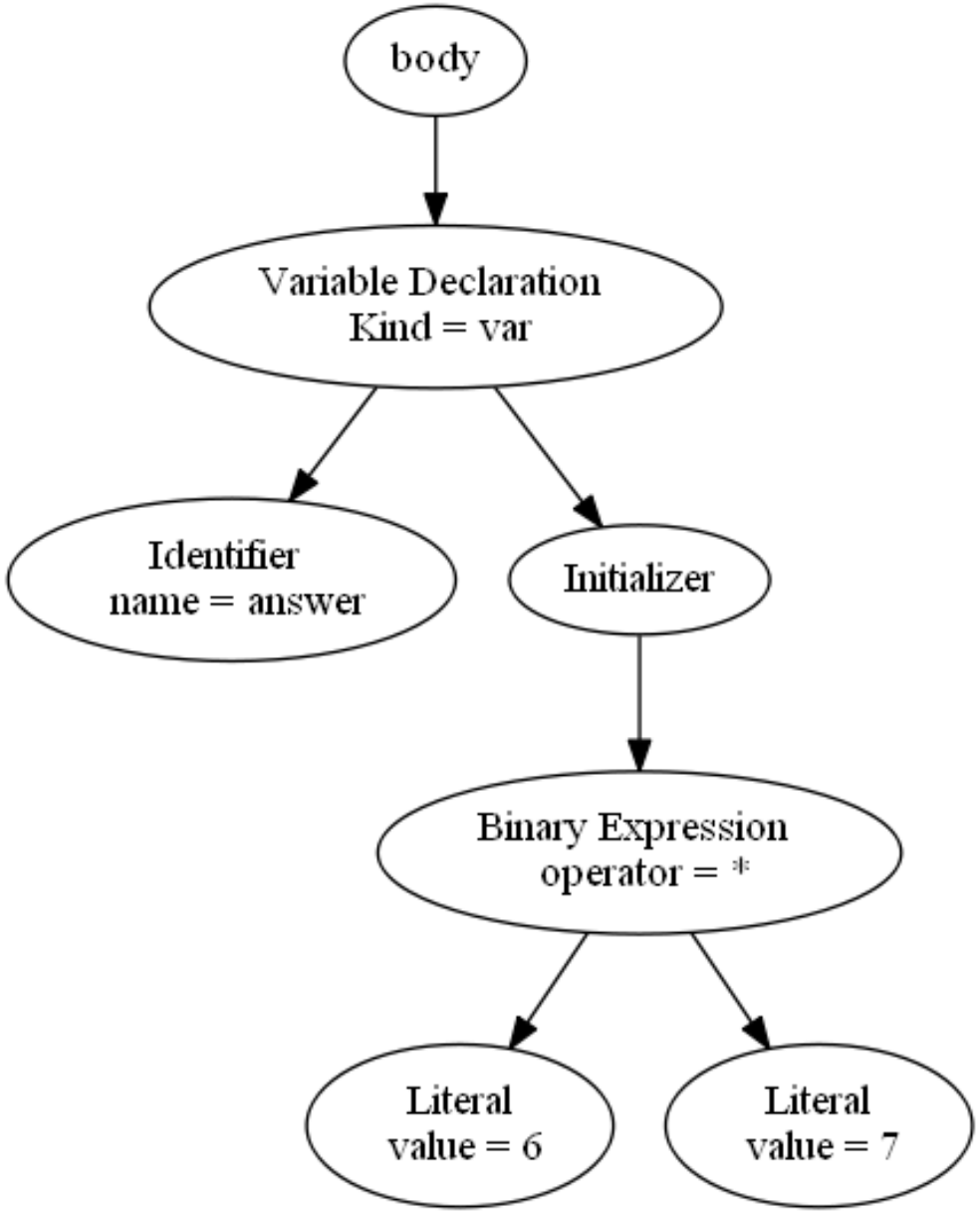
It’s just a function!

VS

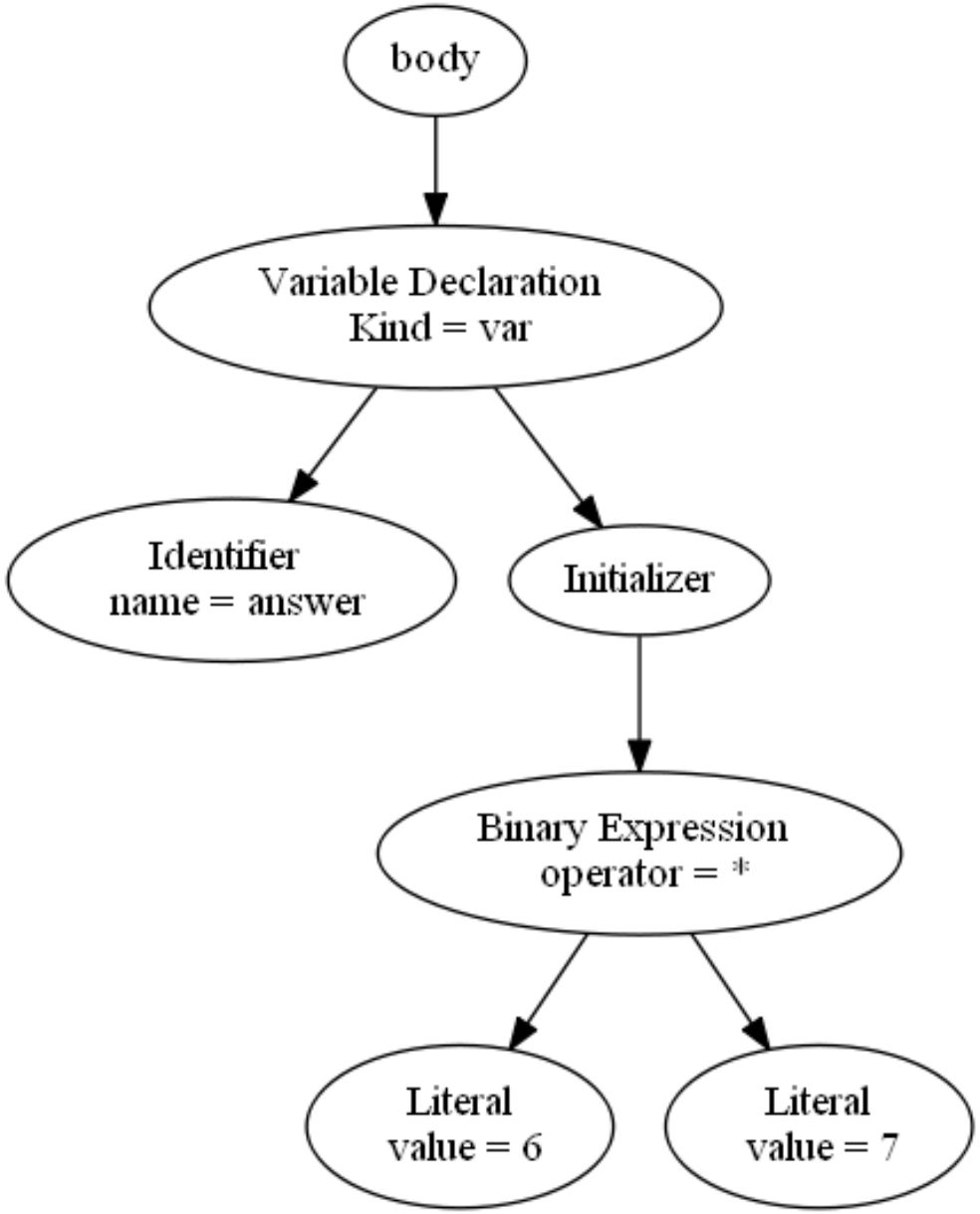
```
def narrate(extractor: ITopicExtractor, outliner IOutliner)(data: AncerstryDataModel): String =
  //Some stuff happens that uses the extractor and outliner
```

Abstract Syntax Trees

```
1  // Life, Universe, and Everything
2  var answer = 6 * 7;
```



```
public interface IJSStatement {}

public class JSProgram : IJSStatement {
    public IJSStatement Body { get; set;}
}

public class VariableDeclaration : IJSStatement {
    //Some enum of kinds. e.g. var, and let.
    public DeclarationKind Kind { get; set; }
    public Identifier Name { get; set; }
    //Null if the variable begins uninitialized
    public IJSStatement Initializer { get; set; }
}

public class BinaryExpression : IJSStatement {
    //Some enum of operators. E.g. *, +, - , / etc.
    public BinaryOperator Operator { get; set; }
    public IJSStatement Left { get; set; }
    public IJSStatement Right { get; set; }
}

public class Literal : IJSStatement {
  public string Value { get; set; }
}
```



```
public interface IJSStatement {
    object Evaluate(ProgramContext context);
}

public class JSProgram : IJSStatement {
    public IJSStatement Body { get; set;}
    object Evaluate(ProgramContext context) { return Body.Evaluate(context); }
}

public class VariableDeclaration : IJSStatement {
    //Some enum of kinds. e.g. var, and let.
    public DeclarationKind Kind { get; set; }
    public Identifier Name { get; set; }
    //Null if the variable begins uninitialized
    public IJSStatement Initializer { get; set; }

    object Evaluate(ProgramContext context) {
        context.symbolTable.defineSymbol(Name.Value,
            Initializer == null ? null : Initilizer.Evaluate(context));
    }
}

public class BinaryExpression : IJSStatement {
    //Some enum of operators. E.g. *, +, - , / etc.
    public BinaryOperator Operator { get; set; }
    public IJSStatement Left { get; set; }
    public IJSStatement Right { get; set; }

    object Evaluate(ProgramContext context) {
        switch(Operator) {
            case BinaryOperator.Multiply:
                return CoerceFloat(Left.Evaluate(context)) * CoerceFloat(Right.Evaluate(context));
            default: throw new NotImplementedException();
    }
}

public class Literal : IJSStatement {
  public string Value { get; set; }

  object Evaluate(ProgramContext context) {
        return Value;
    }
}
```

OOP smears complexity across the data

```
public class Evaluatator {

    public object Evaluate(IJSStatement statement, ProgramContext context) {
        if(statement is JSProgram) return Evaluate( ((JSProgram)statement).Body, context);
        if(statement is VariableDeclaration) {
            var varDec = (VariableDeclaration)statement;
            var value = varDec.Initializer == null ? null : Evaluate(varDec.Initializer, context);
            context.symbolTable.defineSymbol(varDec.Name.Value, value);
            return value;
        }
        if(statement is BinaryExpression) {
            var binExpr = (BinaryExpression)statement;
            switch(binExpr.Operator) {
                case BinaryOperator.Multiply:
                    return CoerceFloat(Evaluate(binExpr.Left,context)) * CoerceFloat(Evaluate(binExpr.Right,context));
                default: throw new NotImplementedException();
            }
        }
        if(statement is Literal) return ((Literal)statement).Value;
        Assert.Fail("Unknown Statement: " + statement);
    }
}
```

```
public class Evaluatator {

    public object Evaluate(IJSStatement statement, ProgramContext context) {
        if(statement is JSProgram) return Evaluate( ((JSProgram)statement).Body, context);
```

```
$('div.active tr.selected').hide()
```

```
                return CoerceFloat(Evaluate(binExpr.Left,context)) * CoerceFloat(Evaluate(binExpr.Right,context));
            default: throw new NotImplementedException();
        }
    }
    if(statement is Literal) return ((Literal)statement).Value;
    Assert.Fail("Unknown Statement: " + statement);
    }
}
```

Pattern Matching

```
type JSStatement =
    | VarDec of VariableDeclaration
    | Id of Identifier
    | BinExpr of BinaryExpression
    | Lit of Literal

and BinaryOperator = MultiplyOp | DivideOp | AddOp | SubtractOp
and DeclarationKind = DecVar | DecLet
and JSProgram = {body:JSStatement}
and Identifier = {name: string}
and VariableDeclaration = {kind:DeclarationKind; id: Identifier; initializer: Option<JSStatement> }
and BinaryExpression = {operator: BinaryOperator; left: JSStatement; right: JSStatement}
and Literal = {value: int}

let rec eval statement (context :System.Collections.Generic.Dictionary<string,string>) =
    match statement with
        | VarDec { kind = DecLet } -> raise (RuntimeError("ES6 Not supported yet"));
        | VarDec { kind = DecVar; id = xId; initializer = xInitializer} ->
            let value = match xInitializer with
                | None -> null
                | Some(statement) -> eval statement context;
            context.Add(xId.name,value);
            value;
        | BinExpr {operator = xOperator; left = xLeft; right = xRight} ->
            match xOperator with
                | MultiplyOp -> sprintf "%i" (Int32.Parse(eval xLeft context) * Int32.Parse(eval xRight context))
                | DivideOp -> sprintf "%i" (Int32.Parse(eval xLeft context) / Int32.Parse(eval xRight context))
                | AddOp -> sprintf "%i" (Int32.Parse(eval xLeft context) + Int32.Parse(eval xRight context))
                | SubtractOp -> sprintf "%i" (Int32.Parse(eval xLeft context) - Int32.Parse(eval xRight context))
        | Lit {value=xValue}-> sprintf "%i" xValue
        | _ -> null
```

Pattern Matching

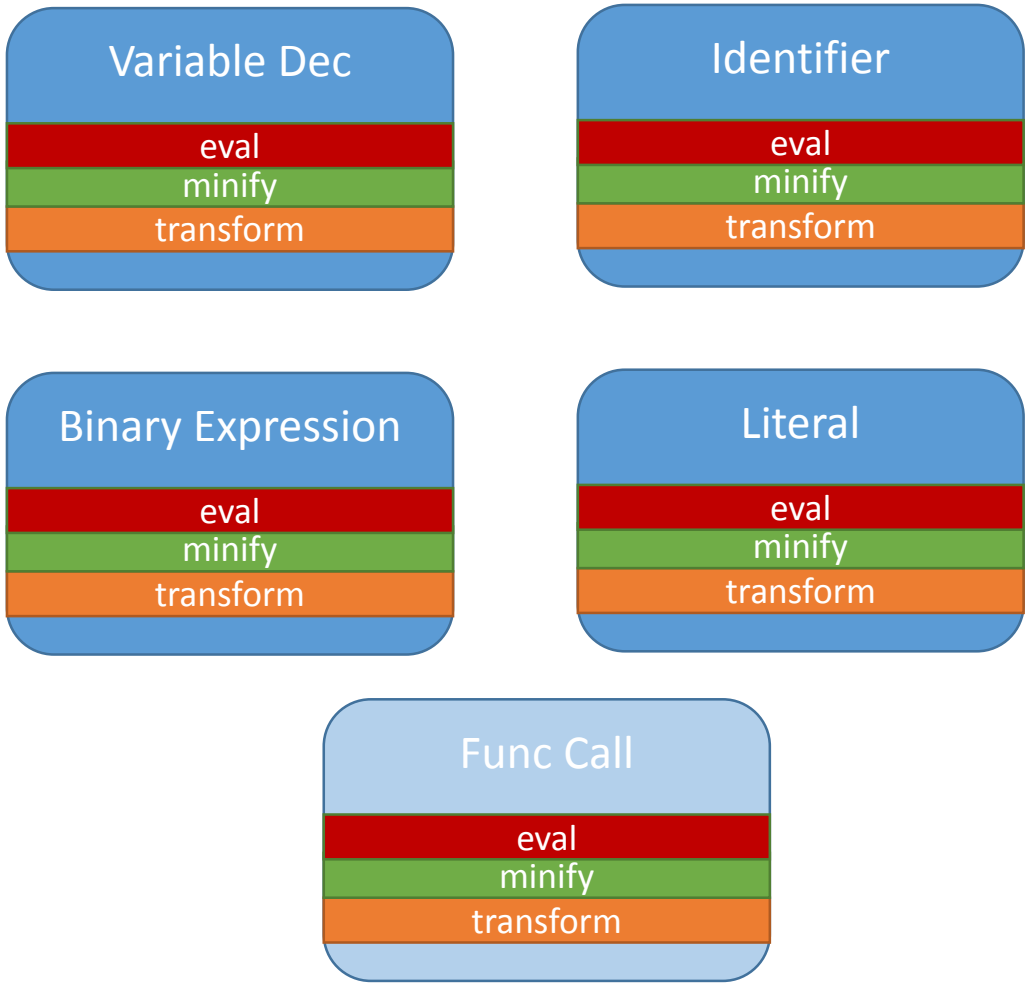
```
type JSStatement =
    | VarDec of VariableDeclaration
    | Id of Identifier
    | BinExpr of BinaryExpression
    | Lit of Literal

and BinaryOperator = MultiplyOp | DivideOp | AddOp | SubtractOp
and DeclarationKind = DecVar | DecLet
and JSProgram = {body:JSStatement}
and Identifier = {name: string}
and VariableDeclarati
and BinaryExpression
and Literal = {value:
```

A really elegant switch statement

```
let rec eval statement (context :System.Collections.Generic.Dictionary<string,string>) =
    match statement with
        | VarDec { kind = DecLet } -> raise (RuntimeError("ES6 Not supported yet"));
        | VarDec { kind = DecVar; id = xId; initializer = xInitializer} ->
            let value = match xInitializer with
                | None -> null
                | Some(statement) -> eval statement context;
            context.Add(xId.name,value);
            value;
        | BinExpr {operator = xOperator; left = xLeft; right = xRight} ->
            match xOperator with
                | MultiplyOp -> sprintf "%i" (Int32.Parse(eval xLeft context) * Int32.Parse(eval xRight context))
                | DivideOp -> sprintf "%i" (Int32.Parse(eval xLeft context) / Int32.Parse(eval xRight context))
                | AddOp -> sprintf "%i" (Int32.Parse(eval xLeft context) + Int32.Parse(eval xRight context))
                | SubtractOp -> sprintf "%i" (Int32.Parse(eval xLeft context) - Int32.Parse(eval xRight context))
        | Lit {value=xValue}-> sprintf "%i" xValue
        | _ -> null
```

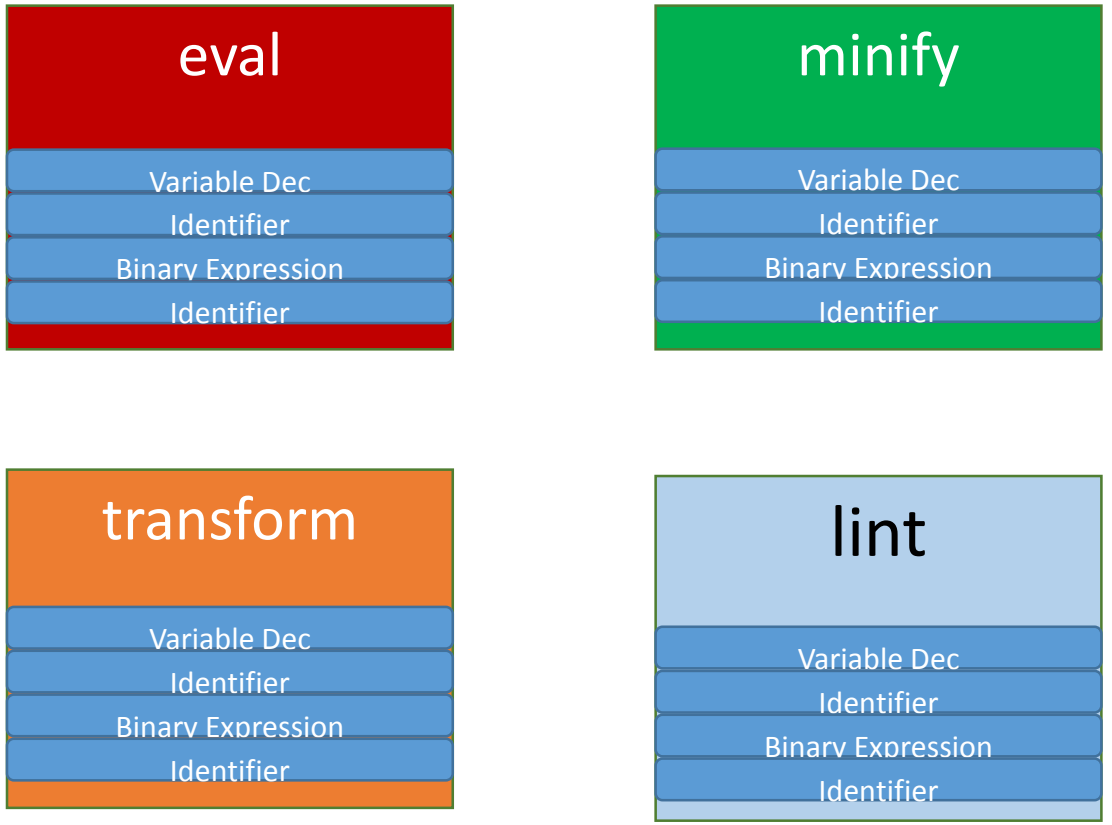
OOP

Coupling: High
Cohesion: Low



Algorithm Centric

Coupling: Medium to High
Cohesion: High

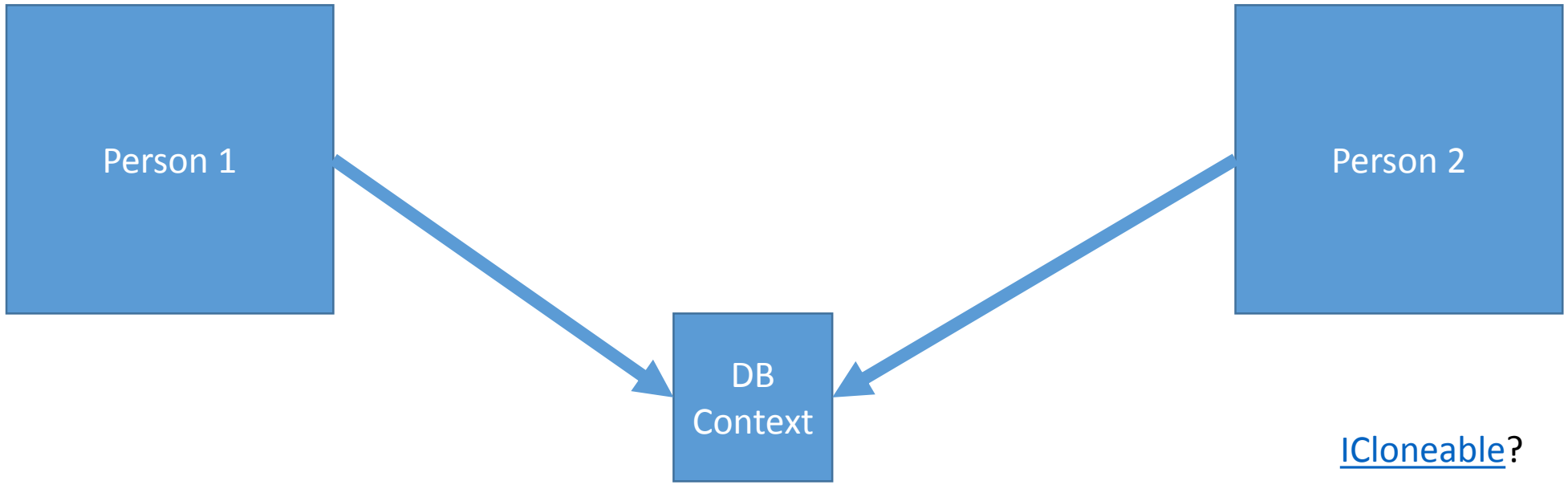


Cloning

```
type MyRecord = { value1: string; value2: string }
let myRec1 = { value1="1"; value2="2" }
let myRec2 = { myRec1 with value2 = "3"}
```

Cloning

```
type MyRecord = { value1: string; value2: string }
let myRec1 = { value1="1"; value2="2" }
let myRec2 = { myRec1 with value2 = "3"}
```



ICloneable?

List Monad

```
Func<Point,int,int,IEnumerable<Point>>  findNeighbors =
    (Point p, int maxX, int maxY) =>
        new []{ -1, 0, 1}
        .SelectMany(xOffset => new []{-1, 0, 1}.Select(yOffset => new Point{ X = xOffset, Y = yOffset}))
        .Where(op => op.X != 0 || op.Y != 0)
        .Select(offset => new Point { X = p.X + offset.X, Y = p.Y + offset.Y})
        .Where(np => np.X >= 0 && np.Y >= 0 && np.X <= maxX && np.Y <= maxY)
        .OrderBy(x => x.X)
        .ThenBy(x => x.Y)

        ;
```

So what are my other tools?

- Partial Function Application
- Pattern Matching
- Cloning
- List Monad (LINQ or Lodash)
- Reactive Programming
- Sets (SQL)
- Etc…