```
Outline of NASSLLI course.
  Sources:
    NYU course wiki:
      http://lambda.jimpryor.net
    Github repositttory for ESSLLI course taught by CB and Dylan:
      https://github.com/dylnb/esslli2015-monads


  CB & JP:
  Introduction
  Big Picture and goals
    Theoretical computer science ==> {philosophy, linguistics}
      Monads ==> compositional semantics that tracks...
        ...intensionality (cf. Winter and Ben-Avi
        ...binding
        ...mutability
        ...meta discourse
      Modularity, factoring, clean design,
      Attitude, not a technique or program---
        a way of abstractly describing some things you already do
    Specific goal for this course: re-engineering of a fragment due to
      Groenendijk, Stokhof and Veltman for dynamic semantics with "might"
    Plan:
      Day 1: Types
      Day 2: Functional programming
      Day 3: GSV straight up
      Day 4: monads
      Day 5: GSV monadicized

Day 1: Introduction, declarative model of computationa, lambda, simple types, system F

  JP: declarative model of computation vs. sequence of directives

    2 + 3 < 7
    2 + 3
    2
    regex patter == set of strings that match
    sets of equations in 2 vars == set of assignments that make it true

  CB:
    Predicate Calculus
      Var = x | y | z
      Cons = a | b | c
      Pred = left | slept
      Rel = saw | loved
      Ind = Var | Cons  ; traditionally "Term"; need "term" for lambda
      For = Pred Ind | Rel Ind Ind | For and For | For or For | not For | \exists For |
\forall For

    Types: Ind has type e             individuals
           For has type t              truth values
           Pred has type e -> t        fn from ind to tv (set of ind)
           Rel has type e -> (e -> t)  fn from ind to e -> t

    lambda calculus
      http://lambda.jimpryor.net/topics/week2_lambda_intro/

    Term =   Var                ; variable      VALUE
           | (\lambda Var Term) ; abstract      VALUE
           | (Term Term)        ; application   PROGRAM
```

```
    x, \x.x, \x.y, \x\y.x, \x.(y (\x.x)), \x.xx

    Details: Lucas Champollion tutorital on the Lambda Calculator
             7:15 Monday evening, Milledoler 100

    Beta reduction: ((\lambda Var Body) Arg) ~~> Body{Var <- Arg}

    ((\x.x) x)
    ((\x.y) x)
    ((\x.y) y)
    (((\x\y.xy) x) y)
    (((\x\y.yx) x) y)

    [alpha reduction, variable collision: see Champollion tutorial Mon 7pm]

    simply-typed lambda calculus
      http://lambda.jimpryor.net/topics/week5_simply_typed/

    Types: Type = e | t | Type -> Type

           Var can have any type
           (\lambda a b) has type a -> b      fn; set of ordered pairs
           (a->b   a) has type b              fn/arg application

    [which examples can't be typed?]

    strongly normalizing (with a view towards black-box side effects)
      Proof: www.mpi-sws.org/~dg/teaching/pt2012/sn.pdf
    denotational semantics with functions as sets of ordered pairs

  JP:
    polymorphism: identity functions
    conjunction = product types?
    Apply type operators
    ...

Day 2: Functional programming

Day 3: GSV straight up
    Reading: Groenendijk, Stokhof, and Veltman, "Coreference and Modality" (1996)
      http://lambda.jimpryor.net/readings/coreference-and-modality.pdf
    NYU course:
      http://lambda.jimpryor.net/topics/week10_gsv/
    ESSLLI course:
      https://github.com/dylnb/esslli2015-monads/tree/master/gsv
    Notes on GSV, with links to code
      http://lambda.jimpryor.net/topics/week10_gsv/

Day 4: monads
  Safe division, intensionalization, ?mutability
  Ken Shan Monads for natural language semantics (2001) uses reader monad to implement
intensionality.
    http://arxiv.org/abs/cs/0205026v1
  Ben-Avi and Winter A modular approach to intensionality (2007) reinvents the technique.
    http://parles.upf.es/glif/pub/sub11/individual/bena_wint.pdf

  monad stacking, monad transformers

Day 5: GSV re-engineered with monads
```