COMP 302 Lecture 14 17 October 2016

Core concepts (recap):

- Eval everything recursively
  - eval(AST node, environment) -> string
    - text -> text
  - definitions
    - eval the name and parameters names
    - record that (do not eval the body)
  - invocations
    - look up the invocation name (eval args and name first)
    - create a new environment with param:arg bindings
    - exec the body in that environment
  - parameters {{{…}}}
    - look that up in the environment we just create
    - if we don't find it -> bound to itself (in the sense of the syntactic version of the parameter: "{{{…}}}")

With this, we can already do interesting things. We can do nested calls.
Ordering is implicit *see 14c302*

We can pass functions around (in a limited way) *see 14c302-1*

Recursive functions *see 14c302-2*

Arithmetic and Numbers

- Could define numbers and operators inside our grammar.
- We know we're "living on top of JS"
- We will assume that there is an already defined expr grammar {{ #expr | …}}
  - The above is a special, always-recognized template
  - Its body is an arithmetic expression.
  - #expr evaluates its argument and return the result *see 14c302-3*
  - N.B. this is a glaring security hole since the input can be anything (should sanitize it)
  - Can also include logical operators

Conditionals

- *See 14c302-4*
- Notice "true" for us is a non-empty string, and "false" as an empty string.
- We need to treat #if and #ifeq differently:
  - eval the name
  - eval the condition
  - then we choose to evaluate the then or else, we only do one of them.

See 14c302-5

Scoping and environment
See 14c302-6

- Notice that we will need to keep track of the declared environment for templates
- For static scoping, this should be the declared environment ->

bar: param -> …
body ->
environment ->

Executing in an environment E0