Scheme Semantics

Douglas Blank

Lambda

Lambda

- We've seen that lambda + define = Python's def

```
def function(x):
    return x + 1

(define function
    (lambda (x)
        (+ x 1)))
```

Lambda

- There is no “return” statement in Scheme
- All expressions evaluate to a value
  - The last expression in a lambda-body is the value for the entire expression
- “define” is used with procedures and other types, such as numbers, strings, etc.
- “define” and “lambda” are separate and independent

Lambda

- (lambda (x) x)
  #<procedure>
- ((lambda (x) x) 1)
- What is a good name for (lambda (x) x)?
- (define function
    (lambda (x)
      (lambda (y) (< y x))))
- *Where you create the procedure may be a different place than where you invoke it*

Lambda

- Anonymous function
- Python actually has lambda (well, a limited one)
- func = lambda x: x + 1
- Python's lambda is limited to a single expression (not statements)
- Scheme does not make a distinction between statements and expressions

Let vs. Lambda

- (let ((x 2) (y 5))
    (+ x y))
- ((lambda (x y) (+ x y)) 2 5)
- *let* can be implemented with *lambda*

Let & Lambda

- Let and lambda create local variables
- These exist only in the body of the lambda or let
- Their *scope* is only in that expression
- When you invoke a lambda, it binds values to the local variables
- If a variable is not local, then it is *free*
- Lambda's can be nested

Closure

- When you create a procedure with lambda, it actually captures all of the local variables and their values
- This is called an *envionment*
- If you invoke a closure somewhere else, it still has access to the environment it was created in
- To find the value of a local variable, Scheme has to search through the nested environments