Today

- Homework
  - Mon: Mini-project 2 due Thursday at 11:59 pm
- let
- lambda

Treating functions as things

- “define” associates a name with a value
  - The usual form associates a name with an object that is a function
    ```scheme
    (define (square x) (* x x))
    (define (pi) 3.1415926535)
    ```
  - You can define other objects, though:
    ```scheme
    (define *pi* 3.1415926535)
    (define *month-names* '(january february march april may june july august september october november december))
    ```

"Global variables"

- Functions are “global”, in that they can be used anywhere:
  ```scheme
  (define (pi) 3.1415926535)
  (define (circle-area radius) (* (pi) radius radius))
  ```
- A "global" variable, similarly, can be used anywhere:
  ```scheme
  (define *pi* 3.1415926535)
  (define (circle-area radius) (* *pi* radius radius))
  ```

Are these the same?

Consider two forms of "month-name"

```scheme
(define (month-name1 date)
  (first date))

(define (month-name2 first)
  (define month-name2 first))
```

Let

```scheme
(let
  ((variable1 value1) ;;definition 1
   (variable2 value2) ;;definition 2
  )
  statement1 ;;body
  statement2 ... )
```
Using \texttt{let} to define temporary variables

- \texttt{let} lets you define variables \texttt{within} a procedure:

\begin{verbatim}
(define (scramble-523 wd)
  (let ((second (first (bf wd)))
        (third (first (bf (bf wd))))
        (fifth (item 5 wd)))
    (word fifth second third))

(scramble-523 'neaty) \Rightarrow \texttt{yee}
\end{verbatim}

Three ways to define a variable

1. In a procedure call (e.g., the variable \texttt{proc}):

\begin{verbatim}
(define (doit proc value)
  ;; proc is a procedure here...
  (proc value))
\end{verbatim}

2. As a global variable

\begin{verbatim}
(define *alphabet* '(a b c d e ...))
(define *month-name* '(january ...))
\end{verbatim}

3. With \texttt{let}

Which pi?

\begin{verbatim}
(define (square x)
  (* pi pi))

(define pi 3.1415)
(define (square x)
  (* pi pi))

(define pi 3.1415)
(define (square pi)
  (* pi pi))
\end{verbatim}

Anonymous functions: using \texttt{lambda}
the lambda form

- "lambda" is a special form that returns a function:

\[ \text{lambda } (\text{arg1 arg2 ...}) \]

\[ \text{statements} \]

(\text{lambda } (x) \ (* x x))

a procedure that takes one argument and multiplies it by itself

Using lambda with define

- These are the same:

\[ \text{define (square x)} \]

\[ (* x x) \]

\[ \text{(define square} \] \)

\[ \text{(lambda } (x) \ (* x x)) \]

Using lambda with define

- These are VERY DIFFERENT:

\[ \text{define (adder-1 y)} \]

\[ \text{(lambda } (x) \ (+ x 1)) \]

\[ \text{(define adder-2} \]

\[ \text{(lambda } (x) \ (+ x 1)) \]

Accumulate

\[ \text{(define (my-accum1 accum-proc num-sent)} \]

\[ \text{(if (= (count num-sent) 1)} \]

\[ \text{(first num-sent)} \]

\[ \text{(accum-proc} \]

\[ \text{(first num-sent)} \]

\[ \text{(my-accum1 accum-proc (bf num-sent)) } \) \)

\[ > \text{(my-accum1 '245)} \]

\[ \text{(- 2 (- 4 5))} \]

\[ \Rightarrow 3 \]