# Input/Output terminology for procedures

<table>
<thead>
<tr>
<th>“input”</th>
<th>“output”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“take”</td>
<td>“yield”</td>
</tr>
<tr>
<td>“consume”</td>
<td>“produce”</td>
</tr>
<tr>
<td></td>
<td>“return”</td>
</tr>
</tbody>
</table>

“The procedure takes a list and returns a list.”

“The procedure consumes two lists and outputs a number.”
Comments on previous lecture

Prof. Hughes used square brackets [ ] in writing a cond expression. These are the same as ( ). We don’t use them in Scheme this year but you are welcome to.

John used succ? and pred in writing recursive procedures that operate on natural numbers.

succ? and pred are not built-in procedures.
In previous year, they were provided.

Quiz 1: Write succ? and pred
Difference between denotation and evaluation

A text might denote a data object.
Denotations never change.

The evalpig will try to eat any data object you can feed it.
Whether it succeeds and what it yields depends on the context.
Let the Rulz be your guide.

What is the text that denotes the Boolean value true?
possibilities:
• #true
• #true
• true

Built-in procedure to test if a data object is a symbol:
(symbol? ...)

(symbol? #true) → (symbol? (quote #true))
(symbol? true) → (symbol? (quote true))
Built-in procedure to test if a data object is a Boolean:

(boolean? ...)

(boolean? (quote #true))
(boolean? (quote true))

Only one text denoting true?

(symbol? (quote #t))
(boolean? (quote #t))
**parity**

**input:** natural number \(n\)

**output:** the symbol `odd` if \(n\) is odd, the symbol `even` if \(n\) is even.

**type signature:**
\[\mathbb{N} \rightarrow \{\text{odd, even}\}\]

(\text{check-expect} \ (\text{parity} \ 7) \ (\text{quote} \ \text{odd}))
(\text{check-expect} \ (\text{parity} \ 4) \ (\text{quote} \ \text{even}))

Using built-in procedures `odd?` and `even?`?

(\text{define} \ \text{parity} \n  \ (\lambda (n) \n    ((\text{odd?} \ n) \ (\text{quote} \ \text{odd})) \n    ((\text{even?} \ n) \ (\text{quote} \ \text{even}))))

Let's pretend we didn't have `odd?` and `even?` available to us. There is another good method, using arithmetic operators. Just for kicks, I'm going to write it the wrong way---recursively---instead.
parity

input: natural number \( n \)
output: the symbol odd if \( n \) is odd, the symbol even if \( n \) is even.

type signature:
\[ \mathbb{N} \rightarrow \{ \text{odd, even} \} \]

Recursion diagrams

original input: 17
  recursive input: 16
  recursive output: even
original output: odd

original input: 18
  recursive input: 17
  recursive output: odd
original output: even

Original output is logical opposite of recursive output.

Can use a built-in procedure \texttt{not} that takes a Boolean and returns its logical opposite.
Writing our own version of \texttt{even}?

\begin{verbatim}
(my-even? ...)
\end{verbatim}

\textbf{Input:} integer \textit{n}

\textbf{Output:} true if \textit{n} is even
Is there a doctor in the house?

Procedure `contains-doctor`?

Lists -> Booleans

**input:** a list $L$

**output:** true if $L$ contains the symbol `doctor`
Input a list, output a list

Procedure double

Lists -> Lists

Input: list $L$
Output: a list consisting of the elements of $L$, in the same order, but with each element occurring twice.

e example:
input: (A (B) C)
output: (A A (B) (B) C C)
Innovation Dojo is a semester-long program that places you within a like-minded group of talented students and teaches you to think differently about entrepreneurship. You will learn about all stages of the entrepreneurial process and pursue a project of your own! Info session in Petteruti Lounge from 2:00-3:00 PM on Saturday, September 15. The application deadline is Friday September 21st at 11:59 PM.

APPLY HERE: https://goo.gl/forms/wFQyb74EekDWeqD92.