Types
"Good" vs. "Bad" Expressions

; interp-expr : FAE? ... -> FAE-Value?

• Does interp-expr produce a value for all expressions?

• Of course not!

• (interp-expr (parse '{5 5})) etc ...

• But do we know enough about expressions to tell before actually calling interp-expr?
Quiz

• **Question #1**: What is the value of the following expression?

\[ \{ + \ 1 \ 2 \} \]

• **Answer**: 3
Quiz

• **Question #2:** What is the value of the following expression?

  \{ + \text{fun} \ 17 \ 8 \} \\

• **Wrong answer:** error

• **Answer:** Trick question! \{ + \text{fun} \ 17 \ 8 \} is not an expression
Language Grammar for Quiz

\[
\begin{align*}
\langle \text{MFAE} \rangle &::= \langle \text{num} \rangle \\
&\mid \text{true} \\
&\mid \text{false} \\
&\mid \{ + \langle \text{MFAE} \rangle \langle \text{MFAE} \rangle \} \\
&\mid \{ - \langle \text{MFAE} \rangle \langle \text{MFAE} \rangle \} \\
&\mid \{ = \langle \text{MFAE} \rangle \langle \text{MFAE} \rangle \} \\
&\mid \langle \text{id} \rangle \\
&\mid \{ \text{fun } \{ \langle \text{id} \rangle^* \} \langle \text{MFAE} \rangle \} \\
&\mid \{ \langle \text{MFAE} \rangle \langle \text{MFAE} \rangle^* \} \\
&\mid \{ \text{if } \langle \text{MFAE} \rangle \langle \text{MFAE} \rangle \langle \text{MFAE} \rangle \} 
\end{align*}
\]
Quiz

• **Question #3:** Is the following an expression?

```
{ {{fun {x y} 1} 7}
```

• **Wrong answer:** No

• **Answer:** Yes (according to our grammar)
Quiz

• **Question #4:** What is the value of the following expression?

\[
\{\{\text{fun} \ {x \ y} \ 1\} \ 7\}
\]

• **Answer:** \{\text{fun} \ {y} \ 1\} (according to some interpreters)

• But no *real* language would accept

\[
\{\{\text{fun} \ {x \ y} \ 1\} \ 7\}
\]

• Let’s agree to call \{\{\text{fun} \ {x \ y} \ 1\} \ 7\} an *ill-formed expression* because \{\text{fun} \ {x \ y} \ 1\} should be used only with two arguments

• Let’s agree to never evaluate ill-formed expressions
Quiz

• **Question #5**: What is the value of the following expression?

\[
\{\{\text{fun} \{x \ y\} \ 1\} \ 7\}
\]

• **Answer**: **None** - the expression is ill-formed
Quiz

• **Question #6**: Is the following a well-formed expression?

\[
\{+ \{\text{fun} \{ \} 1\} 8\}\]

• **Answer**: **Yes** (according to our definition of well-formed)
Quiz

• Question #7: What is the value of the following expression?

\[ \{ + \{ \text{fun} \{ \} 1 \} 8 \} \]

• Answer: None - it produces an error:

   \textit{numeric operation expected number}

• Let’s agree that a \texttt{fun} expression cannot be inside a + form
Quiz

• **Question #8:** Is the following a well-formed expression?

```
{ + { fun { } 1 } 8 }
```

• **Answer:** **No** (according to our new definition)
Quiz

• Question #9: Is the following a well-formed expression?

\[ \{+\{\{\text{fun}\{x\}\ x}\ 7\}\ 5\} \]

• Answer: Depends on what we meant by *inside* in our most recent agreement
  
  ○ *Anywhere inside* - No
  
  ○ *Immediately inside* - Yes

• Since our interpreter produces 12, and since that result makes sense, let’s agree on *immediately inside*
Quiz

• **Question #10**: Is the following a well-formed expression?

```
{+  {{fun {x} x} {fun {y} y}} 5}
```

• **Answer**: Yes, but we don’t want it to be!
Quiz

• **Question #11**: Is it possible to define **well-formed** (as a decidable property) so that we reject all expressions that produce errors?

• **Answer**: **Yes**: reject all expressions!
Quiz

• **Question #12:** Is it possible to define *well-formed* (as a decidable property) so that we reject *only* expressions that produce errors?

• **Answer:** *No*

```
{ + 1 {if ... 1 {fun {x} x}} }
```

• If we always knew whether . . . produces true or false, we could solve the halting problem.
Types

• Solution to our dilemma
  ○ In the process of rejecting expressions that are certainly bad, also reject some expressions that are good

```
{+ 1 {if {prime? 131101}
    1
    {fun {x} x}}}
```

• It’s a tradeoff: do we care more about rejecting bad programs, or about not rejecting good ones?
  ○ Different languages pick different tradeoffs
  ○ **Typed:** Java, Scala, Haskell, etc.
  ○ **Untyped:** Racket, Python, Javascript, etc.
    • AKA dynamically typed
Types

• Overall strategy:
  ◦ Assign a \textit{type} to each expression \textit{without evaluating}
  ◦ Compute the type of a complex expression based on the types of its subexpressions
Types

1 : number

true : boolean

{+ 1 2}

number number

number

{+ 1 false}

number boolean

no type