Mystery Language: Function Calls

We don't intend for solving these mysteries to take too long. If you have spent five hours on any part of this assignment please stop and check in with the module instructors or the TAs.

We are developing a new language, one feature at a time. We try multiple prototypes for each feature, but keep losing the source code. Your task is to tell the different variants apart.

Documentation

This documentation includes all of the syntactic constructs we have implemented so far. The latest features to be added or modified are highlighted.

Let

Syntax	**Behavior**
`<name> = <expr>`	Evaluates `expr`, binds the result to `name`. Cannot be the last statement in a block.

Block

Syntax	**Behavior**

`block:`
`  ...`
`end`
	Evaluates each of their statements in order, and evaluates to the value of the final expression in the block.

Parentheses

Syntax	**Behavior**
`(<expr>)`	Parentheses are used for grouping expressions. They do not otherwise change the meaning of the program.

Number Literals

Syntax	**Behavior**
`9000.1` `-42`	Evaluation of a number literal yields a numeric value. Integer and decimal numbers are supported.

Binary Arithmetic Operations

Syntax	**Behavior**
`<expr> + <expr>` `<expr> - <expr>`	These arithmetic operations raise exceptions if their arguments do not evaluate to numbers. Otherwise, they

`<expr> * <expr>` `<expr> / <expr>`	perform the specified operation. (Note: operators like + must be surrounded by whitespace.)

Booleans

Syntax	**Behavior**
`true` `false`	The true and false values of the boolean type.

String Literals

Syntax	**Behavior**
`"I am a string."`	Evaluation of a string literal yields a string value. String values hold ordered sequences of characters. They are usually used as representations of human readable text.

String Concatenation

Syntax	**Behavior**
`<expr> ++ <expr>`	Appends two strings together. This does not modify the strings, but instead produces a new string. (Note: operators must be surrounded by whitespace.)

Comparisons

Syntax	**Behavior**

`<expr> < <expr>`
`<expr> > <expr>`
`<expr> <= <expr>`
`<expr> >= <expr>`
`<expr> == <expr>`
`<expr> != <expr>`
	These operations perform the specified comparison, returning a boolean. Only numbers can be compared. (Note: operators must be surrounded by whitespace.)

Conditionals

Syntax	**Behavior**

`if <condition>:`
`  <then_branch>`
`else:`
`  <else_branch>`
`end`
	Evaluates the `condition`, then evaluates to either the `then_branch` or the `else_branch`, depending on the result of condition.

Function Declaration

Syntax	**Behavior**

`fun <name>(<param>, ...):`
`  <body>`
`end`
	Evaluating a function statement produces a function value and binds it to `name`. The function definition does not evaluate the function body; the evaluation only happens when the function is called. Cannot be the last statement in a block.

Function Application

Syntax	**Behavior**
`<expr>(<expr>, ...)`	Applies a function to zero or more arguments

Assignment Statement

Syntax	**Behavior**
`<name> := <expr>`	Evaluates `expr`, and changes the value that `name` is bound to.

Today's Feature: Function Calls

Focus on how functions call one another, and also themselves.

We coded up a few prototypes in hand-tuned assembly. They may be unintelligible, but wow are they fast!

Setup

To install the languages, follow the installation guide.

The language variants for this assignment are:

`#lang FunctionCalls1` (Core)
`#lang FunctionCalls2` (Core)
`#lang FunctionCalls3` (Core)

To run them all at once, use:

`#lang FunctionCalls`

Submission

The version of Racket and the Mystery Languages should be 7.0.

Submit your classifiers and explanation via the corresponding Moodle form.