

Mystery Language: Mutable Structures

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

`<name> = <expr>`

Behavior

Evaluates `expr`, binds the result to `name`. Cannot be the last statement in a block.

Block

Syntax

`block:`
 `...`
`end`

Behavior

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

Parentheses

Syntax

`(<expr>)`

Behavior

Parentheses are used for grouping expressions. They do not otherwise change the meaning of the program.

Number Literals

Syntax

`9000.1`
`-42`

Behavior

Evaluation of a number literal yields a numeric value. Integer and decimal numbers are supported.

Binary Arithmetic Operations

Syntax

`<expr> + <expr>`
`<expr> - <expr>`

Behavior

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

`true`
`false`

Behavior

The true and false values of the boolean type.

String Literals

Syntax

`"I am a string."`

Behavior

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

`<expr> ++ <expr>`

Behavior

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

`<expr> < <expr>`
`<expr> > <expr>`
`<expr> <= <expr>`
`<expr> >= <expr>`
`<expr> == <expr>`
`<expr> != <expr>`

Behavior

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

Conditionals

Syntax

```
if <condition>:  
    <then_branch>  
else:  
    <else_branch>  
end
```

Behavior

Evaluates the `condition`, then evaluates to either the `then_branch` or the `else_branch`, depending on the result of condition.

Function Declaration

Syntax

```
fun <name>(<param>, ...):  
  <body>  
end
```

Behavior

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.

Lambda Expression

Syntax

```
lam(<param>, ...): <expr> end
```

Behavior

Produces a first-class function value. It can be applied the same way as named functions.

Function Application

Syntax

```
<expr>(<expr>, ...)
```

Behavior

Applies a function to zero or more arguments

Assignment Statement

Syntax

```
<name> := <expr>
```

Behavior

Evaluates `expr`, and changes the value that `name` is bound to.

Record Literal

Syntax

```
{ <expr>: <expr>, ... }
```

Behavior

On the left of each colon is a field name, and on the right is its definition. This syntax constructs a record where each field name is associated with its corresponding definition.

Field Access

Syntax

```
<expr>[<expr>]
```

Behavior

Evaluates the first expression to a record value and the second expression to a field name, then looks up that field name in that record.

Field Assignment

Syntax

`<expr>[<expr>] := <expr>`

Behavior

The first expression is the record, the second is the field name, and the third is the new value. This syntax changes the record by setting the given field name to the new value.

For instance,

```
record = {"x": 1}
record["x"] := 2
record["x"]
```

will produce 2.

Today's Feature: Mutable Structures

Focus on how compound values (like records) and their fields interact with function applications.

Setup

To install the languages, follow the [installation guide](#).

The language variants for this assignment are:

```
#lang MutableStructures1 (Core)
#lang MutableStructures2 (Core)
#lang MutableStructures3 (Advanced)
```

To run them all at once, use:

```
#lang MutableStructures
```

Submission

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

Submit your classifiers and explanation via the corresponding Moodle form.