IN3040 Obligatory Assignment 1

Release 1.0.0

Cory Balaton

Sep 30, 2022

CONTENTS:

CHAPTER ONE

USAGE

1.1 Requirements

- Python 3.10.X

1.2 Running the tests

Running the tests is very simple. There are 4 test programs that can be run, so you can choose to run them individually by specifying a number 1-4 as the argument, or you can choose to run them all, by specifying "all" as the argument.

Here is an example of how you would run all tests:

```
python tests.py all
```

1.3 Running the interpreter

If you want to test out the interpreter as a whole, you can absolutely do that! There are some test programs in the **robol_programs** directory that have the .robol extension.

Here is an example of how to run the interpreter with a .robol file:

```
./robol robol_programs/loopyloop.robol
```

All the robol programs from the assignment are also there if you want to test them.

CHAPTER TWO

DESIGN

2.1 Robot

This class contains the position, direction, bindings, and statements of the robot.

In addition, it contains something called stack. The stack is available to push values into, so that other classes may retrieve those values in the lifetime of the program. A good example is when an ArithmeticExp interprets a NumberExp, the NumberExp pushes its value to the stack, and then ArithmeticExp pops the stack in order to retrieve that value and use it.

Instead of dividing up bindings, start and statements into separate lists, I decided to put them all into a list called Interpretables (Couldn't come up with a better name), as this makes it easier to create the tests, and there is essentially no difference in how they are handled by the robot.

2.2 References to Robot

most of the classes in robol need a reference to the Robot instance inside the program. The earliest prototype of the language took the robot as a parameter when creating an instance of a class that needed said reference. This created a lot of headaches when it came to making the test programs, so I decided to take away the responsibility from the user by creating a private method in each class that contained class instances that needed references of the robot, and add it to them. This made it a lot more user friendly, and also made it less likely to make mistakes when creating the tests.

2.3 Statements

Statements usually modify the state of the robot, whether it's incrementing/decrementing a binding, turning the robot, or moving the robot. They behave about the same in the sense that they evaluate expressions, and then they modify the state of the robot. Most of the classes here are built the same way, except for Loop, and Stop. Stop just prints out the current position of the robot, but Loop is a bit more interesting.

2.4 Loop

Loop was pretty interesting to make, as you can think of it being like a small program inside the program. Although it doesn't have its own stack and bindings, it does have its own list of statements that work pretty much the same way as the statements list of robot. Where this design lacks a little, is that if you declare any bindings inside the loop, they will be inserted into the robot's bindings dictionary, aka. it will be a global binding, instead of only living in the scope of the loop.

The reason I chose that Loop should have its own list of statements, is because it makes it more organized, so that you don't need to jump back a certain amount of statements of the robot's statements list for each iteration of the loop. This also makes nested loops possible.

2.5 Expression

All expressions work the same. They interpet the expression given, and push the result up to the stack. NumberExp and Identifier directly push the value given to the stack, while ArithmeticExp interprets the left and right side of the equation before performing the operation. Interpreting the left and right side before performing an operation makes NumberExp naturally recursive in nature, so you can nest multiple ArithmeticExp instances inside each other.

2.6 Enums

There are parts of the code where there are comparissons, for example what binary operation is used in an arithmetic expression, and it's natural to express the different choices with enums to make the implementations of other classes more readable.

CHAPTER THREE

INTERFACES

`class robol_lang.interfaces.Expression`

An interface for expressions.

Classes that implement the Expression interface, usually evaluate an expression, and then push that value to the stack of a Robot instance.

`class robol_lang.interfaces.Robol`

This is the interface that most classes inherit from.

All the classes that implement neither a Statement nor an Expression, implement Robol.

`class robol_lang.interfaces.Statement`

An interface for Statements.

Classes that implement the Statement interface, usually modify the state of a Robot instance.

CHAPTER FOUR

ENUMS

class `robol_lang.enums.`**`Assign`**(*value*)
    Signifies if Assignment should increment or decrement.

class `robol_lang.enums.`**`BinaryOp`**(*value*)
    Signifies the binary operation to be used in ArithmeticExp.

class `robol_lang.enums.`**`Direction`**(*value*)
    Signifies the direction Turn should turn.

class `robol_lang.enums.`**`Orientation`**(*value*)
    Signifies the four orientations of the compass

CHAPTER

FIVE

ROBOL

`class robol_lang.robol.Binding(`*ident:* Identifier, *exp:* Expression`)`

Class that contains an identifier and the expression to bind to it.

Attributes: ident (Identifier): The identifier of the binding

exp (Expression): The expression that will be bound to the identifier.

`interpret()`

Binds a value to an identifier.

Interprets ident and exp, and inserts the expression into the bindings dictionary of the robot with the identifier as the key.

Returns: None

`class robol_lang.robol.Grid(`*east:* Expression, *north:* Expression`)`

Class that contains the dimensions of the grid.

This defines the space that the robot is allowed to travel on.

Attributes: east (Expression): How far east the grid goes.

north (Expression): How far north the grid goes.

`interpret()` → None

Interpret both expressions.

The result of the interpretation of the expression will be put on the stack and the class that called this interpret method will then be able to retrieve them by popping the stack.

Returns: None

`class robol_lang.robol.Program(`*grid:* robol_lang.robol.Grid, *robot:* robol_lang.robol.Robot`)`

Class that contains all components necessary to run.

This is the starting point of a robol program, and is necessary in order for a program to work.

Attributes: grid (Grid): The grid that the robot will move on.

robot (Robot): The robot itself.

`interpret()` → None

Calls the interpret method of the robot.

Before calling the interpret method of the robot, the method adds a reference of the program to the robot, and a reference of the robot to the grid.

Returns: None

`class robol_lang.robol.Robot`

Class that interprets interpretables and moves around the grid.

This is the heart of the program. The Robot class holds almost all the information of the program and moves on the grid according to what the interpretables interpret.

Attributes: position (Dict): The current position of the robot.

orientation (Orientation): The current orientation of the robot.

bindings (Dict): The bindings of the robot.

interpretables (List): A list of interpretable instances.

stack (List): The robot's stack to push and pop values.

program (Program): A reference to the program.

`interpret()` → None

Interprets each interpretable in interpretables.

Returns: None

`class robol_lang.robol.Start(`*east:* Expression, *north:* Expression`)`

Class that contains the starting point of the robot.

Attributes: east (Expression): How far east the robot should start.

north (Expression): How far north the robot should start.

`interpret()` → None

Interprets east and west.

This interprets east and west, and inserts the result into the position dictionary of the robot.

Returns: None

CHAPTER SIX

STATEMENTS

class robol_lang.statements.**Assignment**(*identifier:* robol_lang.expressions.Identifier, *assign:* robol_lang.enums.Assign)

Class that will increment or decrement a binding by 1.

Attributes: identifier (Identifier): The identifier to increment/decrement.

assign (Assign): The enum that decides if the identifier should increment or decrement.

interpret() → None

Interprets the identifier and increments/decrements the binding.

Returns: None

class robol_lang.statements.**Loop**

Class that will loop a set of statements until a condition is false

Attributes: statements (Statements): The set of statements inside the loop.

condition (BoolExp): The condition for the loop to continue looping.

interpret() → None

Interprets the statements in the loop.

This interprets the statements of the loop over and over again until the condition is false.

Returns: None

class robol_lang.statements.**Step**(*exp:* Expression)

Class that moves the robot a certain amount of steps.

Attributes: exp (Expression): The expression that evaluates how many steps should be taken.

interpret() → None

Interprets the expression and moves the robot.

First it evaluates the expression, then it checks if the amount of steps would put the robot out of bounds, and if it does, then it raises an exception, otherwise it will move.

Returns: None

class robol_lang.statements.**Stop**

Class that signals that the program is done.

interpret() → None

Prints out the current position of the robot.

Returns: None

class robol_lang.statements.**Turn**(*direction:* robol_lang.enums.Direction)

Class that turns the robot clockwise/counterclockwise.

Attributes: direction (Direction): Which direction to turn.

`interpret()` → None

Sets the new orientation of the robot.

Returns: None

CHAPTER

SEVEN

EXPRESSIONS

class robol_lang.expressions.**ArithmeticExp**(*op:* robol_lang.enums.BinaryOp, *left:* robol_lang.interfaces.Expression, *right:* robol_lang.interfaces.Expression)

Class that evaluates arithmetic expressions

From the assignment, I have interpretted that relational operators are also part of the operators used in arithmetic expressions.

Attributes: op (BinaryOp): The binary operation of the expression.

left (Expression): The left side of the expression.

right (Expression): The right side of the expression.

interpret() → None

Interprets left and right and evaluates the expression.

After interpreting the left and right expression, the method matches the binary operation and performs it with left and right.

Returns: None

class robol_lang.expressions.**BoolExp**(*a_exp:* robol_lang.expressions.ArithmeticExp)

Class that evaluates if an arithmetic expression is true or false.

Attributes: a_exp (ArithmeticExp): The arithmetic expression to evaluate.

interpret() → None

Evaluates the expression to be true or false.

If the arithmetic expression is 0, then the boolean expression is false, otherwise, it's true. It reminds me of how Scheme evaluates if something is true or false.

Returns: None

class robol_lang.expressions.**Identifier**(*identifier: str*)

Class that represents an identifier.

Attributes: identifier (Identifier): The value of the identifier.

interpret() → None

Appends the value to the stack of the robot.

Returns: None

class robol_lang.expressions.**NumberExp**(*val: int*)

Class that represents a number.

Attributes: val (int): The value of the number.

`interpret()` → None

Appends the value to the stack of the robot.

Returns: None

CHAPTER EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX