October 29, 2024

Python Advanced Course

List 4.

Please choose one of the tasks, and if someone chose task 1, you should choose one from variants (A), (B) or (C); it is enough to do one variant. Each task is worth 6 points.

Supplement the implementation with your own exceptions (classes) that react to incorrect situations, e.g. no value assigned to a variable.

Task 1.

Program an Expression class with appropriate subclasses representing different types of arithmetic expressions. For example, the expression

```
(x + 2) * y
```

can be represented as

```
Times(Add(Variable("x"), Constant(2)), Variable("y"))
```

where Times, Variable, or Constant are appropriate subclasses of the Expression class.

Program methods in each class

- calculate(self, variables), which calculates the value of an expression; where the variables argument is a dictionary where the keys are the names of the variables, and values of these variables;
- __str__ returning a nicely formatted expression as a string;
- __add__ and __mul__, which for two objects w1 and w2 of the Expression class creates a new Expression object that represents the sum or product of w1 and w2.

Program the response to incorrect data by throwing appropriate exceptions. For each type of error program your own exception class e.g. VariableNotFoundException.

It is required that constants, variables, and basic arithmetic operations be defined.

Extend the implementation of the above task in one of the following ways.

(A) A similar class hierarchy can be created representing a simple programming language with at least the following statements: assignment statement, conditional if and a while loop statement. A statement representing a sequence of statements will also be useful. It can be assumed that an arithmetic expression equal to we interpret zero as false and true otherwise. In each from these classes you need to program the execute(self, variables) method that executes subsequent instructions.

The __str__ method will also be useful, as it returns a readably formatted program as a string.

(B) Having expressions in the form of such a tree, one can try to simplify the expression, for example expressions containing only constants of the type 2 + 2 + "x" can be simplified to 4 + "x", or you can use the multiplication property by zero. Program at least two such simplifying rules in the form class methods of the Expression class. The simplification operation is to create a new object of class Expression.

(C) We can treat expressions containing only one variable as functions. Program a class method that evaluates an expression (an object of class Expression) that is derived from a function.

Task 2.

Program the Formula class with appropriate subclasses that will represent propositional formulas. For example

¬x ÿ (y ÿ true)

can be presented as

```
Or(Not(Variable("x")), And(Variable("y"), Constant(True)))
```

We assume that formulas consist of the constants True and False, variables, and at least an alternative, conjunction, and negation.

Program methods in each class

- calculate(self, variables), which calculates the value of an expression; where the variables argument is a dictionary, where the keys are the names of variables, and the values are the values of these variables, __str__ returning the formatted expression in infix form as a string;
- __add__ and __mul__, which for two objects w1 and w2 of class Formula create a new Formula object that represents the alternative or conjunction of w1 and w2, respectively.
- .tautology() which checks whether the given formula is a tautology. You can use the itertools package learned in the previous classes.

Program a method (it can be a class method, for example) that calculates a simplification of a given formula using dependencies

p ÿ false ÿ false

Whether

false ÿ p ÿ p

Marcin Mÿotkowski