Advanced Python: Homework set 5

2023/2024

Please choose one of the tasks, and if someone chose task 1, please choose one from variants (A), (B) or (C); just make one variant. Every task is worth 6 points

Problem 1

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

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

Program a 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 are defined.

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

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

The `__str__`- method will also be useful, returning a clearly formatted program as a string.

(B) If you have expressions in the form of such a tree, you 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 property of multiplying by zero. Program at least two such rules to simplify in the form of a method of the class `Expression`;

(C) We can think of expressions containing only one variable as functions. Program a class method that calculates an expression (an object of the `Expression` class) that is a derivative of the function.

Problem 2

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

$$\neg x \vee (y \wedge true)$$

can be presented as Or (Not (Variable("x")), And(Variable("y"), Constant(True)))

Program methods in each class

1. `calculate(self, variables)`, which calculates the value of the expression; where the variables argument is a dictionary, where the keys are the names of the variables and the values are the values of these variables
2. `__str__`- returning a formatted expression in infix form as a string;
3. `__add__` and `__mul__` which, for two objects $w1$ and w2 of the Formula class, creates a new Formula object that represents the disjunction or conjunction of $w1$ and $w2$, respectively.
4. `.tautology()` which checks whether the given formula is a tautology. In practice, it is convenient to program the tautology as a method of `Formula` class.

Program a method (e.g. a class method) that calculates its simplification for a given formula using dependencies

$$p \wedge \text{ false } \equiv \text{ false}$$

Whether

$$\text{false } \vee p \equiv p$$

Program your own exceptions (classes) that respond to incorrect situations, e.g. missing variable value assignment.