ЯТЬ 15

**Условие**

Требуется реализовать набор классов: Scope, FunctionDefinition, Function, Number, Conditional, BinaryOperation, UnaryOperation, FunctionCall, Reference, Print, Read. Классы представляют примитивы некоторого искусственного функционального языка.

В языке есть всего два типа: числа и функции. Почти все конструкции являются выражениями, т. е. возвращают число или функцию. А классы выше представляют абстрактное синтаксическое дерево этого языка (см. <https://en.wikipedia.org/wiki/Abstract_syntax_tree> )

**Описание классов и требования к их интерфейсу:**

***Scope*** - представляет доступ к значениям по именам (к функциям и именованным константам). Scope может иметь родителя, и если поиск по имени в текущем Scope не успешен, то если у Scope есть родитель, то поиск делегируется родителю. Scope должен поддерживать dict-like интерфейс доступа (см. на специальные функции \_\_getitem\_\_ и \_\_setitem\_\_).

Шаблон класса:

class Scope(object):

def \_\_init\_\_(self, parent = None):

pass

Пример использования:

parent = Scope()

parent[“foo”] = Function(...)

parent[“bar”] = Number(10)

scope = Scope(parent)

ans = scope[“bar”] # ans == Number(10)

scope[“bar”] = Number(20)

ans = scope[“bar”] # ans == Number(20)

ans = scope[“foo”] # ans == Function(...)

ans = scope[“zoo”] # undefined behaviour, “zoo” not found

***Number*** - представляет число в программе. **Все числа в нашем языке целые.**

Шаблон класса:

class Number:

def \_\_init\_\_(self, value):

def evaluate(self, scope):

метод evaluate должен возвращать self, т. е. объект класса Number.

***Function*** - представляет функцию в программе. Функция - второй тип поддерживаемый языком. Функции можно передавать в другие функции, и возвращать из функций. Функция состоит из тела и списка имен аргументов. Тело функции это список выражений, т. е. каждого из них есть метод evaluate.

Во время вычисления функции (метод evaluate), все объекты тела функции вычисляются последовательно, и результат вычисления последнего из них является результатом вычисления функции. Список имен аргументов - список имен формальных параметров функции.

Шаблон класса:

class Function:

def \_\_init\_\_(self, args, body):

def evaluate(self, scope):

метод evaluate работает, как описано выше.

***FunctionDefinition*** - представляет определение функции, т. е. связывает некоторое имя с объектом Function. Результатом вычисления FunctionDefeinition является обновление текущего Scope - в него добавляется новое значение типа Function.

Шаблон класса:

class FunctionDefinition:

def \_\_init\_\_(self, name, function):

def evaluate(self, scope):

метод evaluate добавляет объект function под именем name в scope. Пусть метод evaluate возвращает функцию function.

***Conditional*** - представляет ветвление в программе, т. е. if.

Шаблон класса:

class Conditional:

def \_\_init\_\_(self, condtion, if\_true, if\_false = None):

def evaluate(self, scope)

condition - это некоторое выражение, результат вычисления которого обязательно является объектом Number. if\_true и if\_false - списки **(возможно пустые или равные None)** выражений, если результат вычисления condition это объект Number, который хранит 0, то вычисляется if\_false список, иначе if\_true.

Результатом вычисления всего Conditional является результат вычисления последнего элемента в соответствующем (if\_true или if\_false) списке. Если соответствующий список пуст или равен None, то возвращаемое значение остается на ваше усмотрение.

***Print*** - печатает значение выражения на отдельной строке.

Шаблон класса:

class Print:

def \_\_init\_\_(self, expr):

def evaluate(self, scope):

в методе evaluate вычисляется значение выражения expr, который (результат) обязательно является объектом типа Number, и на экран выводится число хранящееся внутри. Вывод завершается переходом на следующую строку (никаких дополнительных символов, лишнего форматирования, научных форматов, отступов и пр). Возвращаемое значение метода evаluate остается на ваше усмотрение.

***Read*** - читает число из стандартного потока ввода и обновляет текущий Scope. Каждое входное число располагается на отдельной строке (никаких пустых строк и лишних символов не будет).

Шаблон класса:

class Read:

def \_\_init\_\_(self, name):

def evaluate(self, scope):

метод evaluate читает со стандартного потока ввода число (на отдельной строке), и добавляет в scope это число под именем name. Возвращаемое значение evaluate остается на ваше усмотрение.

***FunctionCall*** - представляет вызов функции в программе. В результате вызова функции должен создаваться новый объект Scope, являющий дочерним для текущего Scope (т. е. текущий Scope должен стать для него родителем). Новый Scope станет текущим Scope-ом при вычислении тела функции.

Шаблон класса:

class FunctionCall:

def \_\_init\_\_(self, fun\_expr, args):

def evaluate(self, scope):

метод evaluate должен вычислить fun\_expr и результатом этого вычисления будет объект типа Function (назовем его function). Кроме того, он должен вычислить все объекты в списке args, результаты этих вычислений будут позиционными аргументами при вызове функции. Затем метод должен создать новый Scope (назовем его call\_scope), родителем которого является scope. В call\_scope должны быть добавлены результаты вычисления args, под именами указанными в объекте Function в соответствующем порядке. После чего нужно вычислить function с использованием call\_scope, результат этого вычисления будет результатом метода evaluate.

***Reference*** - получение объекта (функции или переменной) по его имени.

Шаблон класса:

class Reference:

def \_\_init\_\_(self, name):

def evaluate(self, scope):

метод evaluate должен найти в scope объект с именем name и вернуть его (см. подробнее про класс Scope).

***BinaryOperation*** - представляет бинарную операция над двумя выражениями. Результатом вычисления бинарной операции является объект Number. Поддерживаемые операции: “+”, “-”, “\*”, “/”, “%”, “==”, “!=”, “<”, “>”, “<=”, “>=”, “&&”, “||”.

Шаблон класса:

class BinaryOperation:

def \_\_init\_\_(self, lhs, op, rhs):

def evaluate(self, scope):

lhs и rhs - левое и правое выражения соответственно; op - строка с обозначением оператора (все допустимые строки приведены выше); метод evaluate должен вычислить значение lhs и rhs, и вернуть Number хранящий значение соответствующей бинарной операции над результатами вычисления lhs и rhs.

Для логических операций и операций сравнения считаем, что Number хранящий 0 соответствует False, а остальные значения соответствуют True.

Гарантируется, что левый и правый операнды (lhs и rhs) при вычислении дадут объект типа Number, т. е. не может получится так, что вам придется, например, сравнивать две функции.

***UnaryOperation*** - представляет унарную операцию над выражением. Результатом вычисления унарной операции является объект Number. Поддерживаемые операции: “-”, “!” **(логическое отрицание, а не факториал)**.

Шаблон класса:

class UnaryOperation:

def \_\_init\_\_(self, op, expr):

def evaluate(self, scope):

метод evaluate должен вычислить expr, и вернуть Number хранящий значение соответствующей унарной операции над результатом вычисления expr. Как и для BinaryOperation, Number хранящий 0 считаем за False, а все остальные за True.

**Замечания:**

1. считайте что в программе, которую нужно вычислить, нет ошибок, т. е. результат вычисления выражения fun\_expr, переданного в качестве параметра конструктору FunctionCall всегда будет объект Function, результатами вычисления lhs и rhs параметров BinaryOperation всегда будет Number, результатом вычисления выражения параметра Print всегда будет Number, все используемые имена всегда объявлены и тд;
2. шаблоны класса и примеры использования описывают минимальные требования к интерфейсу классов, вы можете расширять их по своему усмотрению или определять дополнительные классы не указанные в списке;
3. не изменяйте имена классов и имена, список и порядок параметров и возвращаемые значения обязательных методов;
4. язык функциональный, т. е. функции можно передавать в качестве параметров функций, возвращать из функций, вызывать рекурсивно и тд;
5. функции появляются только в результате исполнения FunctionDefinition, который в свою очередь может находится внутри одной из веток Condtional и пр.

**Общие требования:**

1. Soft дедлайн 7.10.2015, за задания сданные до этого срока вы получаете 5 баллов.
2. Hard дедлайн 14.10.2015, за задания сданные до этого срока вы получаете 2,5 балла.
3. Решение должно представлять из себя один **файл с именем model.py**.
4. В файле model.py должны присутствовать тесты, на которых вы проверяли свое решение, тесты не должны исполнятся при импортировании (import) модуля model.py.