Форматирование кода и частичные вычисления.

В этом задании вам нужно реализовать два класса PrettyPrinter и ConstantFolder. Оба класса должны содержать метод visit с одним параметром - синтаксическим деревом.

PrettyPrinter.

```
class PrettyPrinter:
    def visit(self, tree):
```

в результате работы метода visit, в стандартный поток вывода должна выводится программа на языке ЯТЬ, представленная абстрактным синтаксическим деревом tree (tree - объект какой-либо класса из первого ДЗ). Учтите, что PrettyPrinter создается один раз, а его метод visit может вызываться несколько раз.

Описание синтаксиса языка ЯТЬ

Вся программа в языке ЯТЬ состоит из последовательности предложений (statements). На вход методу visit подается именно одно предложение языка ЯТЬ. После каждого предложения языка и только после них нужно выводит символ “;”. Из чего может состоять предложение языка определено ниже. Обратите внимание, что одна и та же конструкция может в одном контексте являться предложением языка, а в другом нет - будьте внимательны.

Предложение это одно из следующего списка:

1. арифметическое выражение
2. условное выражение (представляется как объект Conditional)
3. определение функции (представляется как объект FunctionDefinition)
4. конструкция print (представляется как объект Print)
5. конструкция read (представляется как объект Read)

Арифметическое выражение это:

1. число (представляется как объект Number)
2. имя (представляется как объект Reference)
3. бинарная операция (представляется как BinaryOperation), аргументами которой являются арифметические выражения
4. унарная операция (представляется как UnaryOperation), аргументом которой является арифметические выражение
5. вызов функции (представляется как FunctionCall)

Таким образом, если известно, что нужно вывести для каждого из использованных в описании выше классов (Conditionl, FunctionDefinition, Print, Read, Number, Reference, BinaryOperation, UnaryOperation и FunctionCall), то синтаксис языка будет полностью задан. Пройдемся по всему списку.

Conditional. Для объекта Conditional необходимо вывести текст следующего вида:

```
if (<condition>) {
    <if true 0>
    <if true 1>
    …
    <if true k>
} else {
    <if false 0>
    <if false 1>
    …
    <if false n>
}
```

где:

1. <condition> - арифметическое выражение, переданное как параметр condition конструктору Conditional
2. <if true 0> - <if true k> - предложения языка (т. е. после них нужно выводить ‘;’), которые переданы как список if_true в конструктор Conditional; учтите, что список if_true может быть пустым, в этом случае нужно вывести просто фигурные скобки
3. <if false 0> - <if false n> - предложения языка (т. е. после них нужно выводить ‘;’), которые переданы как список if_false в конструктор Conditional; учтите, что список if_false может быть None, в этом случае вы должны полностью опустить else часть

Например, следующий код:

```
number = Number(42)
conditional = Conditional(number, [], [])
printer = PrettyPrinter()
printer.visit(conditional)
```

может вывести такой текст:

```
if (42) {
};
```

а может вывести такой:

```
if (42) {
} else {
};
```

FunctionDefinition. Для объекта класса FunctionDefinition необходимо вывести текст следующего вида:

```
def <name>(arg0, arg1, …, argk) {
    <statement 0>;
    <statement 1>;
    …
    <statement n>;
}
```

где:

1. <name> - имя функции, переданное как параметр name в конструктор FunctionDefinition
2. arg0,arg1,...,argk - имена формальных параметров разделенные запятыми (лишних запятых быть не должно), переданные как список args в конструктор Function, которая, в свою очередь, передана как параметр function в конструктор FunctionDefinition; учтите, что список формальных параметров может быть пустым, в этом случае нужно вывести просто круглые скобки
3. <statement 0> - <statement n> - предложения языка (т. е. они должны заканчиваться символом “;”), которые образуют тело функции (параметр body в конструкторе Function); учтите, что тело функции может быть пустым, в этом случае нужно вывести просто фигурные скобки

Например, следующий код:

```
function = Function([], [])
definition = FunctionDefinition(“foo”, function)
printer = PrettyPrinter()
printer.visit(definition)
```

может вывести следующий текст:

```
def foo() {
};
```

Print. Для объекта Print нужно вывести текст следующего вида:

```
print <expression>
```

где <expression> - это выражение, которое передано как параметр expr в конструктор объекта Print.

Например, следующий код:

```
number = Number(42)
print = Print(number)
printer = PrettyPrinter()
printer.visit(print)
```

может вывести такой текст:

```
print 42;
```

Read. Для объекта Read нужно вывести текст следующего вида:

```
read <name>
```

где <name> - это имя переданное как параметр name в конструктор объекта Read.

Например, следующий код:

```
read = Read("x")
printer = PrettyPrinter()
printer.visit(read)
```

может вывести такой текст:

```
read x;
```

Number. Для объекта типа Number нужно вывести число, которое было передано в конструктор в десятичном виде. Например, следующий код:

```
ten = Number(10)
printer = PrettyPrinter()
printer.visit(ten)
```

может вывести такой текст:

```
10;
```

Точка с запятой в конце нужна, так как метод visit принимает на вход предложение языка и поэтому должен вывести точку с запятой после него (в контексте арифметического выражения, например, эта точка с запятой не понадобится).

Reference. Для объекта класса Reference необходимо вывести имя (параметр name) переданное в конструктор Reference. Например, следующий код:

```
reference = Reference("x")
printer = PrettyPrinter()
printer.visit(reference)
```

может вывести следующий текст:

```
x;
```

Точка с запятой нужна в конце по тем же соображениям, что и в случае с Number.

BinaryOperation. Для объекта BinaryOperation нужно вывести левый операнд, который является арифметическим выражением, оператор и затем правый операнд, который так же является арифметическим выражением, с учетом приоритетов (т. е. возможно потребуется вывести дополнительные круглые скобки).

Приоритеты операций и ассоциативность приведены в таблице (как для бинарных операторов так и для унарных), операции в начале таблицы имеют больший приоритет, чем операции в конце таблицы:

Оператор	Ассоциативность
'-' - унарный минус '!' - логическое отрицание	справа-налево
'*', '/', '%'	слева-направо
'-', '+'	слева-направо
'<', '<=', '>', '>='	слева-направо
'==', '!='	слева-направо
'&&'	слева-направо
'||'	слева-направо

Например, следующий код:

```
n0, n1, n2 = Number(1), Number(2), Number(3)
add = BinaryOperation(n1, '+', n2)
mul = BinaryOperation(n0, '*', add)
printer = PrettyPrinter()
printer.visit(mul)
```

может вывести следующий текст:

```
1 * (2 + 3);
```

а может вывести такой текст:

```
((1) * ((2) + (3)));
```

Но не может вывести такой текст:

```
1 * 2 + 3;
```

Точка с запятой в конце нужна по тем же соображениям, что и для Number.

UnaryOperation. Вывод для UnaryOperation в целом аналогичен выводу для BinaryOperation, есть только одно замечание - используются префиксные унарные операции.

Например, следующий код:

```
number = Number(42)
unary = UnaryOperation('-', number)
printer = PrettyPrinter()
printer.visit(unary)
```

может вывести следующий текст:

```
-42;
```

а может вывести:

```
(-(42));
```

Точка с запятой в конце нужна по тем же соображениям, что и для Number.

FunctionCall. Для объекта FunctionCall нужно вывести текст следующего вида:

<expr>(<arg 0>, <arg 1>, …, <arg n>)

где:

1. <expr> - арифметическое выражение, переданное в конструктор FunctionCall как аргумент fun_expr
2. <arg 0>, <arg 1>, …, <arg n> - список арифметических выражений, переданные в конструктор FunctionCall как параметр args, все выражения разделены запятыми

(лишних запятых быть не должно); учтите, что args может быть пустым списком, в этом случае нужно вывести пустые круглые скобки

Например, следующий код:

```
reference = Reference("foo")
call = FunctionCall(reference, [Number(1), Number(2), Number(3)])
printer = PrettyPrinter()
printer.visit(call)
```

может вывести следующий текст:

```
foo(1, 2, 3);
```

Требования к реализации PrettyPrinter:

1. в реализации PrettyPrinter вы нигде не должны использовать явную информацию о типах (т. е. функции isinstance, type и им подобные конструкции языка python, которые позволяют явно проверять тип объекта)
2. лишние пробелы и лишние скобки игнорируются
3. **вывод должен содержать разумные отступы** - синтаксис языка этого не требует, но мы это будем проверять (т. е. тела функций и списки предложений в Condtional должны иметь больший отступ, чем окружающий их контекст)
4. для отступов используйте либо пробелы либо табуляции, но не мешайте оба символа вместе.
5. обратите внимание, что точка с запятой ставится только после предложений языка

ConstantFolder.

```
class ConstantFolder:
    def visit(self, tree):
```

метод visit класса ConstantFolder должен возвращать новое синтаксическое дерево, построенное на основе параметра tree такое, чтобы в нем не осталось операций вида:

- BinaryOperation(Number, AnyBinOp, Number), где Number - какой-либо объект класса Number, а AnyBinOp - любая бинарная операция из списка допустимых бинарных операций;
- UnaryOperation(AnyUnOp, Number), где AnyUnOp - любая операция из списка допустимых унарных операций;
- BinaryOperation(Number(0), '*', Reference), где Number(0) - объект класса Number, который содержит значение 0, Reference - любой объект класса Reference;
- BinaryOperation(Reference, '*', Number(0));
- BinaryOperation(Reference(name), '-', Reference(name)), где Reference(name) - объект класса Reference, содержащий имя name.

но при этом семантически “эквивалентное” исходному дереву tree.

Общие требования:

1. работа сдается в виде трех файлов: model.py - модуль содержащий измененное описание классов из предыдущего задания, printer.py - содержит реализацию класса PrettyPrinter и folder.py - содержит реализацию класса ConstantFolder
2. если вам нужно импортировать model.py в одном из ваших файлов используйте префикс yat, т. е. например, так: import yat.model (model.py можно положить в папку yat).
3. soft дедлайн для 1-2 групп 15.10.2017, для 3 группы 22.10.2017, задания сданные к этому сроку оцениваются в 5 баллов
4. hard дедлайн для 1-2 групп 22.10.2017, для 3 группы 29.10.2017, задания сданные к этому сроку оцениваются в 2,5 балла.
5. запрещается использовать любые встроенные в Python конструкции, позволяющие явно определить тип объекта: type(foo), foo.__class__, isinstance и прочие; используйте паттерн “Visitor”. Разрешается их использовать только в visit_binary_operation и visit_unary_operation у ConstantFolder.