Team Proposal

{Py}{Thon}

COMS 4115 Spring 2021

February 3rd, 2021

Team members
Cameron Miller (cm3959)
Daniel Hanoch (dh2964)
Gabriel Clinger (gc2821)
George DiNicola (gd2581)

1 Overview of {Py}{Thon}

{Py}{Thon} is a language inspired by Python, but without the constraint of indentation to indicate blocks. {Py}{Thon} retains the syntax that makes Python an easy and fun language, while enforcing curly braces to indicate blocks and semicolons to indicate statement endings to increase portability. In addition, {Py}{Thon} has the notion of a relation, which is a concept from logic programming. Specifically, our language will attempt to mimic this from the first-order-logic programming language, Prolog.

2 Motivation

The motivation for our language was to create a version of Python that is not bound by indentation rules. For many programmers who use Python, using a Python script on a different machine or deploying it into a production environment can be a nightmare when you receive the error “IndentationError: unindent does not match any outer indentation level”. After this, Python users have no idea if one indentation is broken or all of them. Having to go back into your script and delete until the previous line then press “enter”, then “tab” until your desired definition. We set out to create a version of Python that uses curly braces rather than indentation (like Java or C) to avoid this issue.

3 Language Details

Data types and data structures	Operators	Description	Example
Integers	=, ==, !=, <, <=, >, >=, *, /, +, -, ^	A regular integer type int	x = 6; y = 4 ^ 7;

Booleans	=, ==, !=, and, or	Evaluates to true/false	rainy = true;
			wet = false;
			rainy and wet;
			(evaluates to false)

Floats	=, ==, !=, <, <=, >, >=, *, **, /, +, -, ^	A double-precision float type	0.3 + (⅛) (evaluates to 0.5)
Strings	=, ==, !=, join	Regular string type. Can also be a character	a = “Colu”; b = “mbia”; join(a,b); (evaluates to “Columbia”)

Lists	In (evaluates to true/false), append	Can hold only homogeneous-type collection	# p = [1,2,3];
			p.append(9);

Relations	is (evaluates to true/false)	A data structure that establishes relationship and returns a boolean for arbitrary queries	r = <bob:loves:mary>; Might implement at a later time

4 Reserved Keywords

print, if/elif/else, for/while, def, in, or/and, return, true/false, is

5 Built-in Functions

- print()
- join()

6 Comments

- Block comments only ## *comment* ##

7 Example Code

7.1 **Loops**

```
for ( e in list) {
        ## This is a block comment ##
        x = “Hi”;
        join(x,e);
}

x = 0
while (x < 10) {
        print(x);
        x++;
}
```

7.2 **if/elif/else**

```
if (x > 100){
        print(“I am larger than 100!”);
} elif (x > 10 and x <= 100) {
        print(“I am smaller than 100, but not 10!”);
} else {
        print(“I am smaller than 10”);
}
```

7.3 **User-defined function**

```
def gcd(x, y) {
        if (y ==0) {
                return x;
        } else {
                return gcd(y, x % y);
        }
}
```

8 Roles and Responsibilities

- Project Manager - Daniel Hanoch
- Language Guru - Gabriel Clinger
- System Architect - Cameron Miller
- Tester - George DiNicola

9 Specifics

- **Language Type:** imperative
- **Type Scope:** static
- **Type Strength:** static (maybe dynamic)
- **Garbage collection:** won't have it
- **Evaluation:** strict
- **Immutability:** mutable
- **Concurrent Programming Option:** No
- **Type Inference:** yes

10 Question for instructors:

- Will there be too much overhead to have type inference?
- Will there be too much overhead to have dynamic type strength?
- Is the relation data structure feasible?