Python++ Language Proposal

Nathan Cuevas `njc2150` Robert Kim `rk3145` Nikhil Min Kovelamudi `nmk2146` David Steiner `ds3816`

January 2021

1 Introduction

Python++ is a general-purpose imperative programming language inspired by Python that aims to create a language as easy to use as Python, but with the added type safety of Java. The language is strongly and statically typed, has no type inferencing, and supports object-oriented programming without requiring that all functions be part of objects.

Python++ uses syntactically significant whitespace as opposed to curly braces and semicolons. In addition to adding static typing to a Python-style syntax, Python++ aims to add quality of life features such as automatic constructor generation for data classes and a new syntax for loops. Due to time limitations, Python++ will not have a garbage collector unless LLVM has a built-in toggle to make it very easy to add.

2 Primitives, Operators, and Data Types

2.1 List of Operators

Python++ has the basic primitives of ints, floats, chars, and booleans. Double and half precision data types like shorts, longs, doubles, etc. are all included as well. Strings are implemented as part of the standard library. Single quotes initialize characters and double quotes initialize strings e.g. 'a' for char, "apple" for string.

If a numeric operation is performed between an integer and a floating point type, the integer will be automatically coerced into a float, as in Python (as opposed to OCaml).

Numeric data types		
short, int, long, float, double	=	Assignment. Example: x = 5
	+	Addition. Example: 1 + 2.2 # returns float 3.2
	-	Subtraction. Example: 2 - 1 # returns int 1
	*	Multiplication. Example: 2 * 2 # returns int 4
	/	Division. Example: 7 / 3 # returns int 2
	%	Modular arithmetic. Example: 7 % 3 # returns int 1
	+=	Assignment operator equivalent to LHS = LHS + RHS. Behaves as in Python. ++ is not supported.
	-=	Assignment operator equivalent to LHS = LHS - RHS. Behaves as in Python. −− is not supported.
	*=	Assignment operator equivalent to LHS = LHS * RHS. Behaves as in Python.
	/=	Assignment operator equivalent to LHS = LHS / RHS. Behaves as in Python.
	==	Comparison. Checks for equality.
	>	Comparison. Greater than operator.
	<	Comparison. Less than operator.
	>=	Comparison. Greater than equals.
	<=	Comparison. Less than equals.
Booleans		
boolean	=	Assignment.
	==	Checks for equality.
	!=	Not equals.
	not	Negation.
	or	Disjunction.
	and	Conjunction.
Text		
char, string	=	Assignment.
	==	Checks for equality.
	+	Concatenate. For two chars, create a string as in ‘a’ + ‘b’ # return “ab”. For strings, “ab” + “cd” # return “abcd”.

2.2 Arrays

Lists in Python++ are structured similarly to Python lists, but require explicitly defined types and lengths. Arrays are zero-indexed.

```
int[6] first_6_odds = [1, 3, 5, 7, 9, 11]
char[6] name = ['N', 'I', 'K', 'H', 'I', 'L']
string string_name = "david"
println(name) # outputs "NIKHIL" with standard print library
println(first_6_odds[2]) # outputs "5"
```

2.3 Dictionaries

Dictionaries and other data structures will not be built-in for the language, and instead will be offered as part of the standard library.

2.4 Keywords

The following keywords are reserved for the compiler and cannot be used as identifiers: **short int long float double boolean char string def class construct return returns loop while if elif else true false void self required optional static null**

3 Loops and Conditionals

3.1 Universal Loop

Python++ uses a novel loop syntax different from languages we are familiar with. There is no “for” keyword, only “loop” and “while”.

```
# Example of our loop as a while loop:
int y = 0
int x = 5
loop while x >= 0:
    y += 1
    x -= 1
println(y)
println(x)
# when loop exits, y should contain 6 and x should contain -1.

# Example of our loop as a for loop:
int iterations = 0
int i = 0
loop i += 1 while i < 10:
    iterations += 1
println(iterations)
# loop should go through 10 iterations
```

3.2 Conditional Statements

```
# Example of conditional with elif statement:
boolean x = true
boolean y = true
if x and y:
    println("x and y are both true")
elif x or y:
    println("either x or y are true")
else:
    println("neither x nor y are true")
```

3.3 Comments

Single-line comments are indicated by inserting a ‘#’ character at the beginning of text string.

```
# Example comment
int num = 65
```

Multi-line comments are started by string “/#” and end with string “#/”

```
/# Example comment
Example comment, cont.
Example comment, cont.
Example comment, cont.
#/
int num = 65
```

4 Functions

Python++ supports function declarations in a style that is a mix of Python and Java, with a novel syntax for the return type. If the return type is void, specifying the type is optional.

```
def foo(int x):
    x = x + 1
    return
```

is equivalent to

```
def foo(int x) returns void:
    x = x + 1
    return
```

Functions which return values must have a type specified for the compiler in the following way.

```
def foo(int x, int y) returns string:
    if x and y:
        return "x and y are both true"
    elif x or y:
        return "either x or y are true"
    else:
        return "neither x nor y are true"
```

5 Object Oriented Programming

5.1 Class Structure

Python++ has support for basic classes, but will not have support for inheritance or abstract classes. A novel syntax is introduced to separate instance variables from static variables. Another improvement over Python is that methods on classes do not need to take “self” as an argument.

```
class MyClass:
    static:
        # Indicates the variables in this block are static variables.
        int PI = 3

    required:
        # All non-static variables are instance variables.
        # See section on autogenerated constructors for the difference between required and optional.
        int x
        int y

    optional:
        # ALL optional variables must be defined here with default values,
        # but the default value is allowed to be null.
        string foo = "foo"
        string bar = "bar"

    def construct(string x, string y): # optional constructor, see next section for details.
        self.x = (int) x
        self.y = (int) y

    def foo(int x, int y) returns string:
        if x and y:
            return "x and y are both true"
        elif x or y:
            return "either x or y are true"
        else:
            return "neither x nor y are true"
```

5.2 Autogenerated Constructors

Constructors can be automatically generated based on the “required” and “optional” groupings given above. The compiler will automatically generate the following constructors.

```
class MyClass:
    static:
        # Indicates the variables in this block are static variables.
        int PI = 3

    required:
        # All non-static variables are instance variables.
        int x
        int y

    optional:
        # ALL optional variables must be defined here with default values,
        # but the default value is allowed to be null.
        string foo = "foo"
        string bar = "bar"

    def construct(int x, int y):
        # The user did not write this constructor.
        # This constructor is automatically generated by the compiler.
        self.x = x
        self.y = y
        self.foo = "foo"
        self.bar = "bar"

    def construct(int x, int y, string foo, string bar):
        # The user did not write this constructor.
        # This constructor is automatically generated by the compiler.
        self.x = x
        self.y = y
        self.foo = foo
        self.bar = bar
```

Due to the language being statically typed and the time limitation of the project, we will only support these two constructors. That is to say, we will not allow the user to pass in a kwargs dictionary. So the user will either have to specify all of the optional instance variables or none of them.

5.3 Object Casting and Operator Definitions

Python++ allows for user defined typecasting as syntactic sugar to make the language safe while retaining some advantages of weakly typed languages. This typecasting is programmer-definable and can be used between class instances. If a typecast definition is not defined, the compiler throws an error. This typecasting is meant to be very powerful but also potentially dangerous. It is meant to be used when absolutely sure that a said typecast is sufficient for the class to function as expected.

```
class MyOtherClass:
    required:
        int a
        int b

    def cast() returns MyClass: # cast definition
        return MyClass(self.a, self.b)

class MyClass:
    required:
        int x
        int y

    def cast() returns MyOtherClass: # cast definition
        return MyOtherClass(self.x, self.y)

MyClass instA = MyClass(1, 3)
MyOtherClass instB = (MyOtherClass) instA
```

This is syntactic sugar that the compiler will convert to

```
MyClass instA = MyClass(1, 3)
MyOtherClass instB = instA.cast()
```

Similarly, our language also allows definitions for how operators should work with objects. If an operator function is not defined, the compiler will throw an error when it is used on the object.

```
class MyClass:
    required:
        int x
        int y

    def _+(MyClass b) returns MyClass: # addition function
        return MyClass(self.x+b.x, self.y+b.y)

    def _-(MyClass b) returns MyClass: # subtraction function
        return MyClass(self.x-b.x, self.y-b.y)

    def _%%%(MyClass b) returns int:
        # any combination of symbols prefixed with an underscore can be used
        return self.x * b.x

MyClass instA = MyClass(1, 3)
MyClass instB = MyClass(2, 2)
MyClass sum = instA + instB # compiler will convert this to instA._+(instB)
MyClass difference = instA - instB # compiler will convert this to instA._-(instB)
int num = instA %%% instB  # compiler will convert this to instA._%%%(instB)
println(sum.x) # prints '3'
println(sum.y) # prints '5'
println(difference.x) # prints '-1'
println(difference.y) # prints '1'
println(num) # prints '2'
```

5.4 Scoping

To keep things simple, all variables outside of function declarations will be treated as global variables regardless of class hierarchy. The only notion of variable scope will be when the compiler decides what variables to include in the auto constructor for each class.

6 Memory Management

The language will not have any pointers exposed to the user. Whether the language will have automatic garbage collection will depend on whether LLVM makes it very easy to add a garbage collector or not. Unless LLVM includes it as a toggle, we do not plan to have garbage collection in our language.

7 Standard Library

Data structures such as dictionaries (hash maps), as well as mathematical functions such as raising a number to a certain power, are not built in as keywords but will rather be implemented in the Python++ language as part of the standard library. As part of the project, we will implement a few data structures, mathematical functions, and algorithms in the language.