# Scolkam

Programming Language Proposal
COMS W4115 Programming Languages and Translators
Prof. Stephen Edwards
February 19, 2016


| Steve K. Cheruiyot | Yekaterina Fomina | Connor P. Hailey | Léopold Mebazaa | Megan O'Neill |
|---|---|---|---|---|
| (skc2143) | (yf2222) | (cph2117) | (lm3037) | (mo2638) |
| System Architect | Language Guru | System Architect | Manager | Tester |


## Summary


The Scolkam programming language is a subset of the Python programming language that compiles down to LLVM. Scolkam takes the most basic features of Python (arithmetic operations, control flow, etc.) and adds some of the object-oriented features (class definition, class functions, etc.). This language is mostly a technical exercise.


## Language Features

### Code-Block Detection

Unlike Python, the beginning and the end of a function or a branch would not be defined by indentation. We would compel users to write *end* to show the end of code-block (as shown below).

```
def happyBirthday():
    print("Happy Birthday to Scolkam!")
end
```

### Comments

Inline and multi-line comments would stay the same as in Python.

```
a = 0.0 # This is a comment
x = 3
''' This is a multi-line comment
z = 3   '''
y = 3
# a, x, y are defined
# z is not defined
```

### Data Types

As in Python, all data is represented by an object. These objects are immutable. We are not sure of the architecture of data storage, but we are not planning to make any distinction between some kind of "primitive" data types and objects.

- *NoneType*: A constant that would hold the value *None*, and that would be used when a item is undefined. The complete behavior of that type has yet to be defined. Notably, we still have to determine whether users are able to define objects with a undefined value.
- *Numeric types*: *int* for signed integer values, *bool*, to hold *True* or *False* values, and *float* for signed floating-point values. The length of these types would probably be 1 or 2 bytes. (Python's integers have unlimited precision[1], so this will be a departure from the original language.)
- *Sequence types*: *tuple* to store immutably a list of data objects.
- *Text sequence types*: *str* to store a list of characters. Minimally, we would allow for ASCII support. If time permits, Unicode support will be added.

### Static Typing

In contrast to Python, our language would be statically typed. Users would have to initiate their variables when they declare it. We would infer the type of the variable at declaration. This would be an equivalent of putting a C++ *auto* identifier in front of every variable declaration.

### Operations

- *Truth Value Testing*: *and*, *or*, *not*
- *Comparisons*:
  - == and ! =, to check the equivalence of two objects (same value),
  - *is not* and *is*, to check the equality of two objects (same memory address),
  - <, <=, >, >= to rank objects
- *Arithmetic*:
  - +, −, ∗, /, % — integer and floating-point operations

### Control Flow

- Branches: *if*, *else if*, *else*
- Loops
  - *while* — Behave like one would expects, repeating the loop as long as the condition is still met
  - *for* — This loop would be implemented differently than in Python. Since we do not have native range data structures, if a user wants to loop over a range of number, the syntax will be `for i in 1..10` instead of `for i in range (1, 10)`. A user can also iterate on a tuple.

[1] URL: https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

### Class and object management

Python has a significant object management portion. It would therefore be useful to implement some parts of it. Here are multiple existing Python features that could be implemented. These features are ranked in the order of difficulty, and, in general, except for inheritance, we assume that a feature cannot be implemented without having the features listed directly above in this list.

- *Attributes*: The ability to create a class with attributes. (For instance, a 3D vector would have attributes *x*, *y*, *z*.) Adding attributes on the fly, outside of the scope of the class definition, would be a good contribution to the language.
- *Initializer*: The ability for users to write *__init__* functions. We would implement normal initializers (*__init__()*). Copy initializers (*__init__(other-object)*) **would be a potential add-on**.
- *Object Functions*: The ability for users to write functions that are attached to an instance. **Function decorators would be a potential add-on feature.** Here are some that we are thinking to potentially implement:
  - *@classmethod* (for shared methods that have the instance as the first argument)
  - *@staticmethod* (for methods that do not have the class or the instance as first argument),
  - *@property* (for read-only methods),
  - *@[attr-name].setter* (for methods that set the value of a specific attribute),
  - *@[attr-name].deleter* (for methods that delete a specific attribute)
- *Inheritance*: Similar to Python, inheritance would be defined by subclassing classes. All classes would be derived from a special top-level parent class.
- *User defined default functions*: The ability for users, as in C++, to modify some functions that are common to each members, such as the addition operator (*__add__*), the conversion to boolean (*__bool__*), iterator functions, etc. That would require us to not only define the functions for default types, but also taking into account cases in which these functions are not defined for some classes. **We consider this feature to be a potential add-on.**

### Misc.

Some other previously undiscussed aspects of the programming language basically copy Python.

- Function declarations follow the same model as Python, with the keyword *def*, with the exception that types are explicitly mentioned. The syntax of type declaration is following the model of Python Static Typing Checker mypy[2]: `def function-name([type: arg-name]-list) -> type`.
- New lines indicate the end of a statement.
- We would give users the ability to print strings, via a *print* statement (as in Python 2) or function (as in Python 3). In the following examples, we assume a Python 2-like statement.

[2] URL: http://mypy-lang.org

- *String formatting.* We were thinking of something under the format:
  `z = "This string is comprised of %s and %s" % (x,y)`
  or something closer to concatenation:
  `z = "This string is comprised of " + x + " and " + y`.
  In the later case, just the implementation of the add operation to strings could be sufficient.

## Sample Examples

### Summation

```
def sum(int: x) -> int:
  if x == 1:
    return x
  else:
    return x + sum(x - 1)
end
```

### GCD

```
def gcd(int: x, int: y) -> int:
  if y == 0:
     return x
  else:
     return gcd(y, x % y)
end
```

### Hello, World

```
def hello() -> None:
  output = "hello, world!"
  print output
end
```

### Iteration

```
numbers = (1,2,3) # Tuple definition
for number in numbers:
  print number
end
```