COMPILER PROJECT PROPOSAL

Ritvik Bhardwaj, Haocheng Hu, and Naaz Sibia

PROJECT DESCRIPTION

SUMMARY

A compiler that takes a subset of **Python3**, and returns a line of code (with the same functionality) that should be *runnable* with **Python3**. Our intention for this compiler is to see how we can exploit Python's current functionality (examples: environments and functional programming) to write programs in one line and use this compiler as an obfuscation tool. We try to achieve this by chaining lambda functions in a *monad* structure, to create more complex programs.

INPUT SPECIFICATION

We define the subset of Python3 that we will compile as follows:

- **Types:** strings, integers, booleans, lists, (potentially) dictionaries.
- **Functions:** arguments (not *args or **kwargs) without input/output operations. The functions should work with the argument and return types mentioned above.
- **Statically-typed input*:** we will only be accepting code that is statically typed. So if `x` is declared as an `int`, it cannot later be assigned to another type.
- **Control flow:** if else statements
- **Loops:** for, while loops

*: Statically typed input is an additional feature, as it will help with ensuring good coding practises, and is not required for our one liner functionality to work.

OUTPUT SPECIFICATION

One line of code with no newlines, no delimiters that convert code to one line (eg. semicolons), and just a high level function, like Racket/Scheme code. This line of code should still be runnable with Python3. In the event that our code is not able to parse the file, it will return a respective error.

EXAMPLE 1: INPUT

Demonstrates if statements, and variables.

```
a = 6
b = 7
if a < b:
    print(a)
else:
    print(b)
```

Example 1: Output

```
(lambda __g: [[(lambda: (print(a)) if (a < b) else (print(b)))()
for __g['b'] in [(7)]] for __g['a'] in [(6)]])(globals())
```

We leverage the use of the globals dictionary to set and use variables without passing an internal state through the function. Likewise, we can use Python's implicit set functionality for it's loops to set and use the value of a & b.

Globals Explanation (how we use and assign variables in our one line code)

```
g = globals()
print(g.get('a', 'does not exist'))
for g['a'] in [1, 2, 3]:
    print(g.get('a', 'does not exist'))
    print(f'the value of "a" is {a}')
```

Output

```
does not exist
1
the value of "a" is 1
2
the value of "a" is 2
3
the value of "a" is 3
```

This example explains how the `globals()` function works. The function provides a reference to Python's global variable environment, and we're able to modify this environment directly via dictionary notation. We're also are able to access variables normally through Python's standard notation. This is a powerful, yet easily abused functionality, and may be seen as bad coding practise, but is a key feature that will help simplify our code (or make it even possible at all). This functionality allows us to set variables in a loop instead of explicitly assigning them.

Example 2: Input

Demonstrates function definitions, for loops, if statements, and function calls.

```
def test(a: int):
    for i in range(5):
        if a  <  7:
            a += i
    return a

print(test(3))
# this outputs 9
```

This is a simple input, we're defining a function that loops over the first 5 natural numbers and returns the smallest sum bigger than 7, given the starting value.

EXAMPLE 2: OUTPUT

```
print((lambda test: test(3))(lambda a: __import__('functools')
.__dict__['reduce']((lambda x, y: x + y if x < 7 else x), range(5), a)))
# this also outputs 9
```

This piece of code is the one line equivalent.

Since this follows a functional language flow, we can read the program from right to left to parse it's functionality, as follows:

- Uses `reduce` (which takes in a reducer function, a list of values and the initial value, and reduces it down into one value by repeatedly calling the reducer function on every value in the list), as the for loop.
- We pass in the function to another function which passes in the starting value 3. With these two points, we're able to create a program that loops over the first 5 naturals and adds to the starting value.