T3 Python

Sanjukta Saha & Zhou Lingxiang
16th April 2020

Agenda

- Overview
- Parsing
- Interpreter
- Demo
- Reach Goals & Further Improvements
- Summary

Overview

- Python: General Purpose, Popular, High Readability, Diverse Extension
- ANTLR4: Powerful Parser Generator, Written in Java
- TypeScript: Typed JavaScript
- Frontend & x-slang

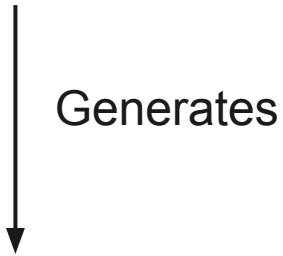
Parsing

Antlr4ts

- Python Grammar File: Python3.g4
- Python3Lexer: Define Key Words, Generate Tokens
- Python3Parser: Nodes Relationship Definition
- Python3Listener: Methods for In and Out Node
- Python3Visitor: Go Through AST (Extended to Generator Classes)

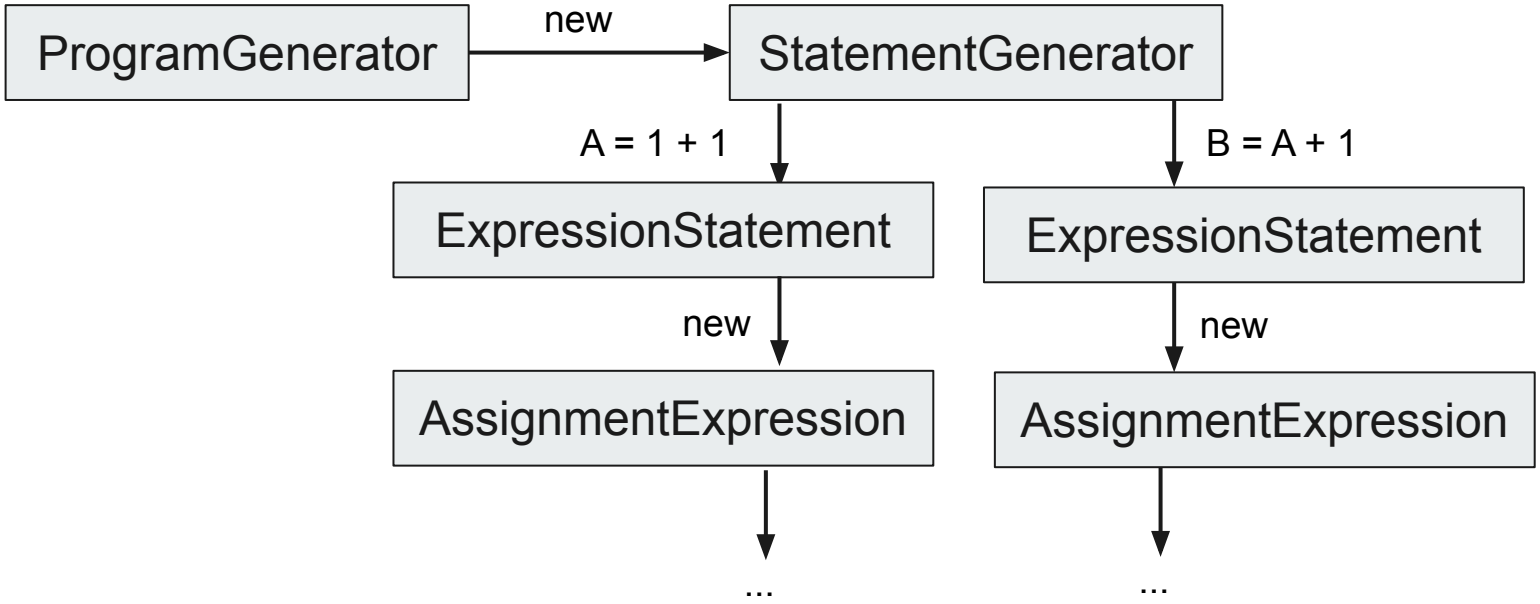
Generator Classes

- ProgramGenerator
- StatementGenerator
- ExpressionGenerator & ExpressionListGenerator



Generator Classes

- A = 1 + 1; B = A + 1



INDENT & DEDENT Tokens

- Python: No Brace Blocks, Separate Block by Indentation
- Python3Lexer: Generate INDENT & DEDENT Tokens
- INDENT Token: Generated after every Compound Statement Colon
- DEDENT Token: Generated after Block Finished

```
def f(a): INDENT
    if a > 0: INDENT
        a = a + 1 DEDENT
    else:     INDENT
        a = a - 1  DEDENT DEDENT
f(3)
```

Algorithm: Recognizing INDENT / DEDENT

- Indentation Stack: IS = [0]
- New Line:
  - Pair braces, brackets, parentheses
  - Get Indentation Number
    - N == head(IS) -> Pass
    - N < head(IS) -> Pop IS M times until N == head(IS), generate M DEDENT tokens
    - N > head(IS) -> Push N into IS, generate INDENT token

a = [1, 3]

```
{
 type: 'Program',
 sourceType: 'script',
 body: [
  {
   type: 'BlockStatement',
   body: [
    {
     type: 'ExpressionStatement',
     expression: {
      type: 'AssignmentExpression',
      operator: '=',
      left: { type: 'Identifier', name: 'a' },
      right: {
       type: 'ArrayExpression',
       elements: [
        { type: 'Literal', value: 1 },
        { type: 'Literal', value: 3 }
       ]
      }
     }
    }
   ]
  }
 ]
}
```

if True:
    pass

```
{
 type: 'Program',
 sourceType: 'script',
 body: [
  {
   type: 'IfStatement',
   test: { type: 'Literal', value: true, raw: 'True' },
   consequent: {
    type: 'BlockStatement',
    body: [
     {
      type: 'BlockStatement',
      body: [ { type: 'PassStatement' } ]
     }
    ]
   },
   alternate: { type: 'EmptyStatement' }
  }
 ]
}
```

Interpreter

- Environment in Python3
- Break / Continue in Loop
- Function Definition & Application
- Global & Nonlocal Statement
- Supported Grammar

The Environment in Python

- Environment Structure: Stack of Frames
- Loops (`for` and `while`) use the *current frame*
- Function applications create and push a *new function body frame* - which treats variables as local variables
    - ... Except when you use **`global`** and **`nonlocal`** keywords (which we discuss in detail later)

Loop

- Use TypeScript `while` loop to evaluate Python `while` loop body
- Use TypeScript `for` loop to evaluate Python `for` loop body
  - The Python for loop is more similar to the forEach function in Typescript
  - The evaluation of for loops in Python involves an iterator, and iterated (e.g. a list or string) and the function body
  - The iterator needs to be assigned a new value with each iteration of the for loop
  - E.g

```
for i in array:
    ...
```

=

```
for iter in iterated:
    ...
```

Break / Continue in Loop

- Break Statement: Generate a **`BreakValue`** Object and Return
- Continue Statement: Generate a **`ContinueValue`** Object and Return
- Block Statement: Return immediately after receiving **`BreakValue`** / **`ContinueValue`** (current cycle stops)
- Use continue / break keywords in TypeScript loop

Function Declaration & Application

- Function Declaration: Assign the whole declaration into an environment frame using the name of the function as the identifier
- Function Application: Get assigned declaration, create and push an empty frame onto environment
- Return Statement: Generate a ReturnValue object with return body and return
- Block Statement: Return immediately after receiving ReturnValue object
- Function Finishes: Check Global list and Nonlocal list, pop frame from

Global and Nonlocal Keywords in Python

Global Keyword

```
c = 1 # global variable

def func():
    c = 2
    print(“In func:”, c)

func()
print(“In main:”, c)

>>> In func: 2
    In main: 1
```

Using Global Keyword

```
c = 1 # global variable

def add():
    global c
    c = c + 2
    print(“In add func:”, c)

add()
print(“In main:”, c)

>>> In add func: 3
    In main: 3
```

Nonlocal Keyword

- Very similar to `global` keyword but primarily used in nested functions

```
x = "global"
def outer():
    x = "local"
    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)

  outer()

print("main:", x)

>>> inner: nonlocal
    outer: nonlocal
    main: global
```

Interpreter logic for `global` keyword Statements

1. Assign variable names in `globallist` as local list variable “global” in current frame
2. Check local variable “global” before function finishes
3. List “global” exists -> Copy the names and their values in list “global” to **Program Frame**
4. List “global” not exists -> Pass

Interpreter logic for `nonlocal` keyword Statements

Similar to the logic for `global` keyword statements

1. Assign variable names in `nonlocallist` as local list variable “nonlocal” in current frame
2. Check local variable “nonlocal” before function finishes
3. List “nonlocal” exists -> Copy the names and their values into **tail frame**
4. List “nonlocal” not exists -> Pass

Supported Grammar

- Unary Expressions
- Binary Expressions (both logic and math operator)
- Variable assignment and variable call
- List and Dict definition and entry access
- Loops (while, for), break and continue in loops
- Conditional Expressions
- Function definition and function application, recursion, return statement
- `global` and `nonlocal` statements

Demo

Recap of our Project Deliverables

Base Level

1. Parser for Python3 using Antlr
2. Implement an interpreter in Typescript, which can deal with basic Python grammars, including the type system, operators, loops, function
3. Implement Python specific features like keywords **global** and **nonlocal**
4. Implement a Python environment visualizer which will be able to construct the data structure figure during processing the program.

Stretch Goals

1. Implement lists, dictionaries and classes
2. Implement some standard library functions (like print, sort, etc.)
3. Improved UI for the environment visualizer

Our Progress

1. Implemented the complete Python3 Parser using Antlr
2. Implement an interpreter in Typescript, which can deal with basic Python grammars, including the type system, operators, loops, function
3. Implemented Python specific keywords global and nonlocal
4. Implemented some of our stretch goals - lists, dictionaries, break, continue

What is pending

1. Integration of the environment model visualiser into our frontend
2. From our stretch goals - implementing some standard library functions, starting with print

We want to implement this by the deadline on next Friday

Q & A