Enhancing Python Syntax: Increment, Decrement, Do- While, Until, Unless & Switch-Case Statements

By: Zak Ahmed, Pedro Oliveira, Piranaven Selvathayabaran

Department of Computing and Software, McMaster University

1280 Main St. W, Hamilton, Ontario, Canada L8S 4L8

April 17th 2019

Problem

One of the most beloved programming languages out there is Python. It contains a plethora of syntax that enables its users to achieve there goals.

However, it does not contain a handful of syntax that we believe would enrich the programming language and provide developers with more flexibility .

The language constructs we believe will ameliorate Python are : **Increment, Decrement, Until , Unless, Do-While Loops, & Switch-Case Statements.** These enhancements, although not necessarily pythonic, would enhance the toolkit of developers.

Process

In order to alter Python, we took advantage of the CPython compiler which has multiple components for compilation which include the following steps[2]:

1. Parse source code into a parse tree (Parser/pgen.c)
2. Transform parse tree into an Abstract Syntax Tree (Python/ast.c)
3. Transform AST into a Control Flow Graph (Python/compile.c)
4. Emit bytecode based on the Control Flow Graph (Python/compile.c)

Several other files needed to be altered to achieve our results. Most notably:

- Python.asdl -> Used for parsing into AST
- Grammar -> Contains grammar rules
- Ceval.c -> executes byte-code
- symtable.c → Generates a symbol table from AST

CPython Modifications & Example Code Additions

GRAMMAR

CPython starts with a grammar, Modifications were done following the grammar notation provided. Similar modifications were made in the Python.asdl , which describes the rules.

```
compound_stmt: if_stmt | unless_stmt | while_stmt | until_stmt | switch_stmt |
unless_stmt: 'unless' test ':' suite('elif test ':')* ['else' ':' suite]
until_stmt: 'until' test ':' suite ['else' ':' suite]
switch_stmt: 'case' test ':' suite('switch' ':' suite)*
factor: ('+'|'-'|'~'|'++'|'--') factor | power
```

AST MODIFICATIONS

Below is an example of the kind of modifications we had to make to allow for 4 different options in a do-while-else loop. Various combinations would allow for various child node configurations which we had to account for, namely:

- While: 4 child nodes
- Do While : 7 nodes
- While, Else : 7 nodes
- Do While Else: 10 nodes

```
static stmt_ty
ast_for_while_stmt(struct compiling *c, const node *n)
{
    /* while_stmt: ['do' ':' suite] 'while' test ':' suite ['else' ':' suite] */
    REQ(n, while_stmt);

    if (NCH(n) == 4) {…
    }
    else if (NCH(n) == 7) {…
    }
    else if  (NCH(n) == 10) {
        expr_ty expression;
        asdl_seq *setup_seq, *seq1, *seq2;

        setup_seq = ast_for_suite(c, CHILD(n, 2));
        if(!setup_seq)
            return  NULL;
        expression = ast_for_expr(c, CHILD(n, 4));
        if (!expression)
            return NULL;
        seq1 = ast_for_suite(c, CHILD(n, 6));
        if (!seq1)
            return NULL;
        seq2 = ast_for_suite(c, CHILD(n, 9));
        if (!seq2)
            return NULL;

        return While(setup_seq, expression, seq1, seq2, LINENO(n), n->n_col_offset, c->c_arena);
    }
```

COMPILING TO BYTECODE

The code on the right demonstrates, the code that had to be created in order to compile the AST to bytecode. The bytecode is handled via several macros. There are also various helper functions like compiler_use_next_block which handles the offsetting of the next instruction.

```
static int
compiler_until(struct compiler *c, stmt_ty s)
{
    basicblock *loop, *end, *anchor = NULL;
    int constant = expr_constant(s->v.Until.test);

    if (constant == 1) {
        return 1;
    }
    loop = compiler_new_block(c);
    end = compiler_new_block(c);
    if (constant == -1) {
        anchor = compiler_new_block(c);
        if (anchor == NULL)
            return 0;
    }
    if (loop == NULL || end == NULL)
        return 0;

    ADDOP_JREL(c, SETUP_LOOP, end);
    compiler_use_next_block(c, loop);
    if (!compiler_push_fblock(c, LOOP, loop))
        return 0;
    if (constant == -1) {
        VISIT(c, expr, s->v.Until.test);
        ADDOP_JABS(c, POP_JUMP_IF_TRUE, anchor);
    }
    VISIT_SEQ(c, stmt, s->v.Until.body);
    ADDOP_JABS(c, JUMP_ABSOLUTE, loop);
```

PARSE TREE

Parsing tokens like increment and decrement required additional changes to tokenizer.c and token.h. We defined them and had to specially handle them since they were a double token. As seen in the switch case statement below, after first checking for the '+' symbol, a following check for the same symbol is done such that the appropriate call is made.

```
    case '+':
        switch (c2) {
        case '+':               return INCREMENT;
        case '=':               return PLUSEQUAL;
        }
        break;
    case '-':
        switch (c2) {
        case '-':               return DECREMENT;
        case '=':               return MINEQUAL;
        case '>':               return RARROW;
        }
        break;
```

Usage Results

Increment

Blurb and photo to be inserted

Decrement

Blurb and photo to be inserted

Until

Blurb and photo to be inserted

Unless

Blurb and photo to be inserted

Do-While

Blurb and photo to be inserted

Switch-Case Statements

Blurb and photo to be inserted

References

[1] Python Software Foundation. (n.d.). 24. Changing CPython's Grammar¶. Retrieved April 09, 2019, from https://devguide.python.org/grammar/

[2] Python Software Foundation. (n.d.). 25. Design of CPython's Compiler¶. Retrieved March 26, 2019, from https://devguide.python.org/compiler/

[3] Python Software Foundation. (n.d.). 10. Full Grammar specification¶. Retrieved April 6, 2019, from https://docs.python.org/3/reference/grammar.html

[4] Python Software Foundation. (n.d.). PEP 3103 -- A Switch/Case Statement. Retrieved March 22, 2019, from https://www.python.org/dev/peps/pep-3103/