

14 Constructing a Tokenizer Level 2

Introduction

```
1 print(-59 + 20*3)
2 a = 2
3 bb_1 = -(a) + 12
4 print(a*bb_1 + a*3*(-1 + -1 + -1))
5
6 # start of code that tests level 2 features
7 print()           # print with no args
8 if a <= 2:         # if, no else
9     if a == 2:     # equal operator
10         if a > 1:  # greater than operator
11             pass   # does nothing
12         print(3)
13 if a != 2:        # if-else
14     print('err#or') # this print should be skipped
15 else:             # else
16     print(4,)      # this print should be executed
17 while a >= 3:      # greater or equal operator
18     print('error') # this print should be skipped
19 a = ----+---5     # multiple unary operators
20 while a < 10:      # while, less than operator
21     print(a)
22     a = +a + .5 + 0.5 # floating-point computation
23 if True:          # True is a boolean constant
24     pass           # pass does nothing
25 print('10\n11')   # escape sequence
26 print('1' + '2')  # string concatenation
27 print(-a + 23)    # unary minus operation
28 if a <= 10:        # less than or equal operator
29     print('#14\'\"'') # pound, quote, backslash in string
30 print(15, 16)      # one space between numbers
31 if (None != False): # None, False, not equal operator
32     print(51/(5 - 2)) # subtraction
```

Figure 14.1

Required Modifications

- No modification required

subtraction operator

- Trivial modification:

pass

if

while

True

False

None

/ (floating-point division)

- Requiring a little thinking

== (equal)

!= (not equal)

< (less than)

<= (less than or equal)

> (greater than)

>= (greater than or equal)

floating-point constants

source code comments

strings delimited by single quotes

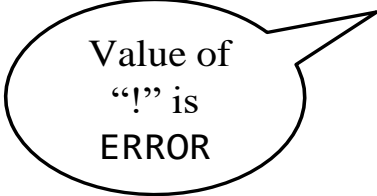
- Requiring serious head scratching

Python indentation

Figure 14.2

Adding Support for Two-character Tokens

```
smalltokens = {'=':ASSIGNOP, '==':EQUAL, '<':LESSTHAN,  
              '<=':LESSEQUAL, '>':GREATERTHAN, '>=':GREATEREQUAL,  
              '!' :ERROR, '!=':NOTEQUAL, '(':LEFTPAREN,  
              ')':RIGHTPAREN, '+':PLUS, '-':MINUS, '*':TIMES,  
              '\n':NEWLINE, '':EOF, ',':COMMA, ':':COLON, '/':DIV}
```



Value of
“!” is
ERROR

```
1 elif curchar in smalltokens:  
2     save = curchar  
3     curchar = getchar() # curchar might be '' (end of source code)  
4     twochar = save + curchar # two chars if curchar != ''  
5     if twochar in smalltokens:  
6         token.category = smalltokens[twochar] # get category  
7         token.lexeme = twochar # get lexeme  
8         curchar = getchar() # move past end of token  
9     else: # one-char token  
10         token.category = smalltokens[save] # get category  
11         token.lexeme = save # get lexeme
```

Adding Support for Floating-Point Constants

Change category if decimal point detected.

Adding Support for Comments

```
c = source[sourceindex]  
sourceindex += 1
```

If `c` is `#` (i.e., the start of a comment), then advance until the newline character is reached.

Adding Support for String Data

Must record that string processing in progress.

Adding Support for Python Indentation

Using indentation

```
1 if a == 1:
2     print('hello')
3     if b == 2:
4         print('goodbye')
5         if c == 3:
6             print('up')
7 print('down')
8 # end of program
```

Using braces (illegal in Python)

```
1 if a == 1:
2 {
3     print('hello')
4     if b == 2:
5     {
6         print('goodbye')
7         if c == 3:
8         {
9             print('up')
10        }
11    }
12 }
13 print('down
```

Figure 14.3

Using indentation

```
1 if a == 1:
2     print('hello')
3     if b == 2:
4         print('goodbye')
5         if c == 3:
6             print('up')
7     print('down')
8 # end of program
```

Using braces (illegal in Python)

```
1 if a == 1:
2 {
3     print('hello')
4     if b == 2:
5     {
6         print('goodbye')
7         if c == 3:
8         {
9             print('up')
10        }
11    }
12    print('down')
13 }
```

Figure 14.4

```
indentstack = [1]
```