Queens College of CUNY
Department of Computer Science
Programming Languages
(CSCI 316)
Winter 2026

Assignment #3
"Lexical and Syntax Analysis"
Due: January 8, 2026

Introduction:

In this first *coding* assignment of our course, we learn about parsing and lexical and syntax analysis, as well as Python packages that support this process.

Submissions:

In the Google form, please submit:

- Assignment03.py (source code)
- Assignment03.txt (console output)

:

Preliminary Tasks:

[1] If you haven't done so already, download and install Python 3.14 and PyCharm Professional IDE (free with your edu email). See the detailed instructions in Assignment #0.

[2] Create a new assignment Assignment03.py.

[3] At the top, add this comment block. (For future assignments, updating the name and number of the assignment.)

```
# Programming Languages (CSCI 316)
# Winter 2026
# Assignment 3 - Lexical and Syntax Analysis
# Jane Doe (your name)
```

[4] At the bottom, add a dummy main() function and the lines that call it:

```
def main():
    pass

if __name__ == "__main__":
    main()
```

Lexical Analysis:

[1] Import these packages

```
import token
import tokenize
from io import BytesIO
```

[2] define a function get_tokens(code):

```
def get_tokens(code):
    return tokenize.tokenize(BytesIO(code).readline)
```

[3] define a function print_tokens_v1(tokens)

```
def print_tokens_v1(tokens):
    for t in tokens
        print(t)
```

[4] define a function print_tokens_v2(tokens)

```
def print_tokens_v2(tokens):
    for t in tokens:
        if t.type in (token.ENCODING, token.ENDMARKER):
            continue
        print(token.tok_name[t.type], repr(t.string), t.start, t.end)
```

[5] Call both versions with various code snippets including

```
code = b"print(1 + 2)\n"

code = b"def add(x, y=2):\n    return x + y"

code = "a = 1 + 2"

code = "def f(x):\n    return x*2\n"
```

and one more of your choosing

Syntax Analysis:

[1] Import this additional package

```
import ast
```

[2] Define a function dump_parse_tree(code)

```
def dump_parse_tree(code):
    tree = ast.parse(code)
    print(ast.dump(tree, indent=2))
```

[3] Define a function walk_parse_tree(code)

```
def walk_parse_tree(code):
    for node in ast.walk(tree):
        print(type(node).__name__)
```

[4] Call both versions with the various code snippets under Lexical Analysis

The parsing expects normal string text while the lexical analyzer expects binary. Convert as follows:

```
code_str = code_bin.decode()
code_bin = code_str.encode()
```

Integration:

[1] Combine the lexical and syntax analysis into one master function:

```
def analyze(code):
    print("=== TOKENS ===")
    tokens = get_tokens(code)
    for t in tokens:
        if t.type in (token.ENCODING, token.ENDMARKER):
            continue
        print(f"{token.tok_name[t.type]:<12} {t.string!r:<10} {t.start}->{t.end}")

    print("\n=== AST ===")
    tree = ast.parse(code)
    print(ast.dump(tree, indent=2))
```

[2] Call analyze for all the code snippets above