VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Name: Apurba Koirala

Reg no: 22BCE3799

Subject Code: BCSE307P

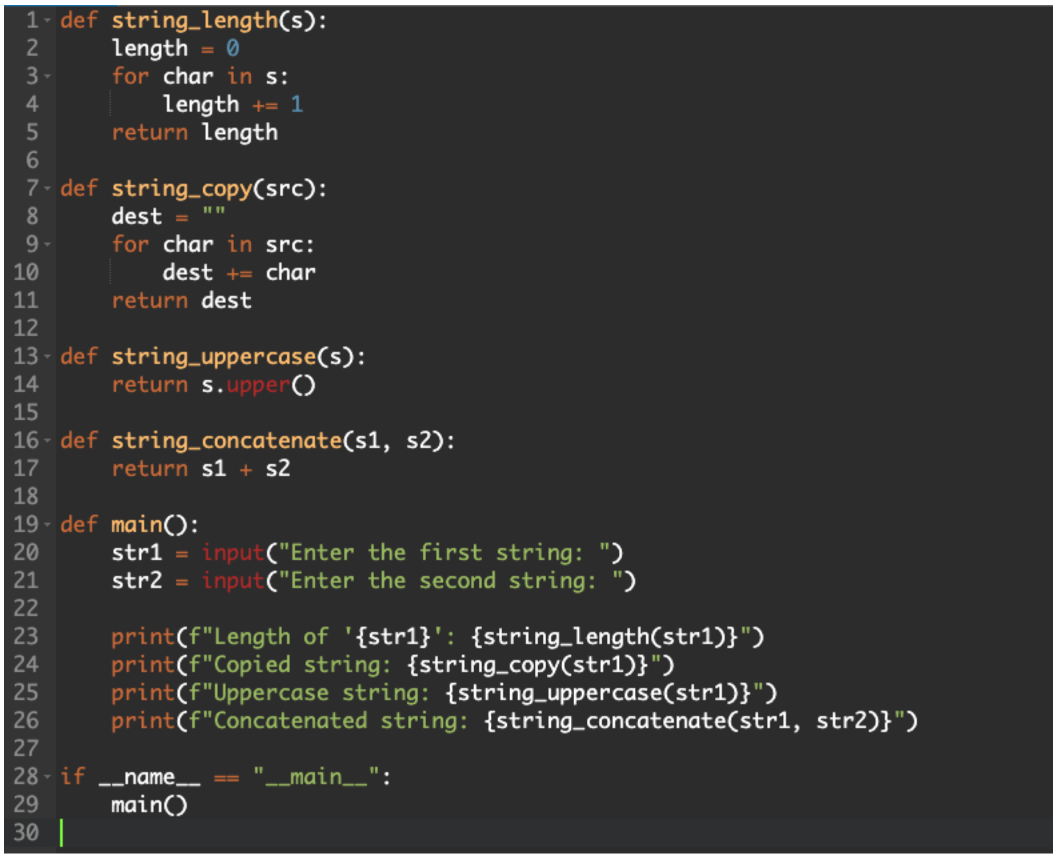
Course Title: Compiler Design Lab

Lab Slot: L49 + L50

Guided By: Dr. Kannadasan R Lab

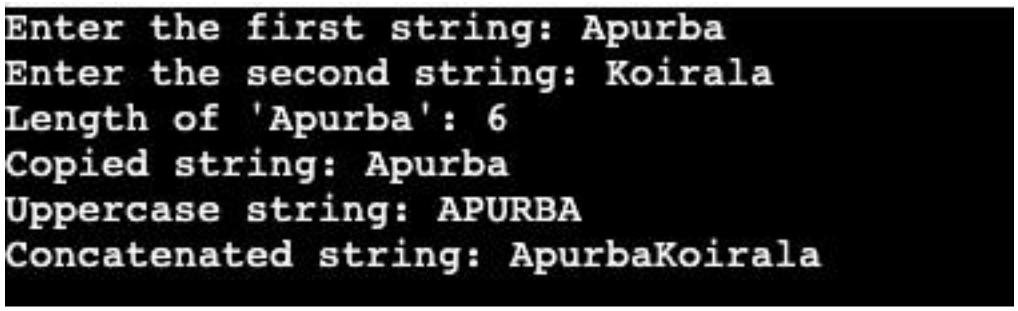
Assessment 1.

String manipulation in Python:



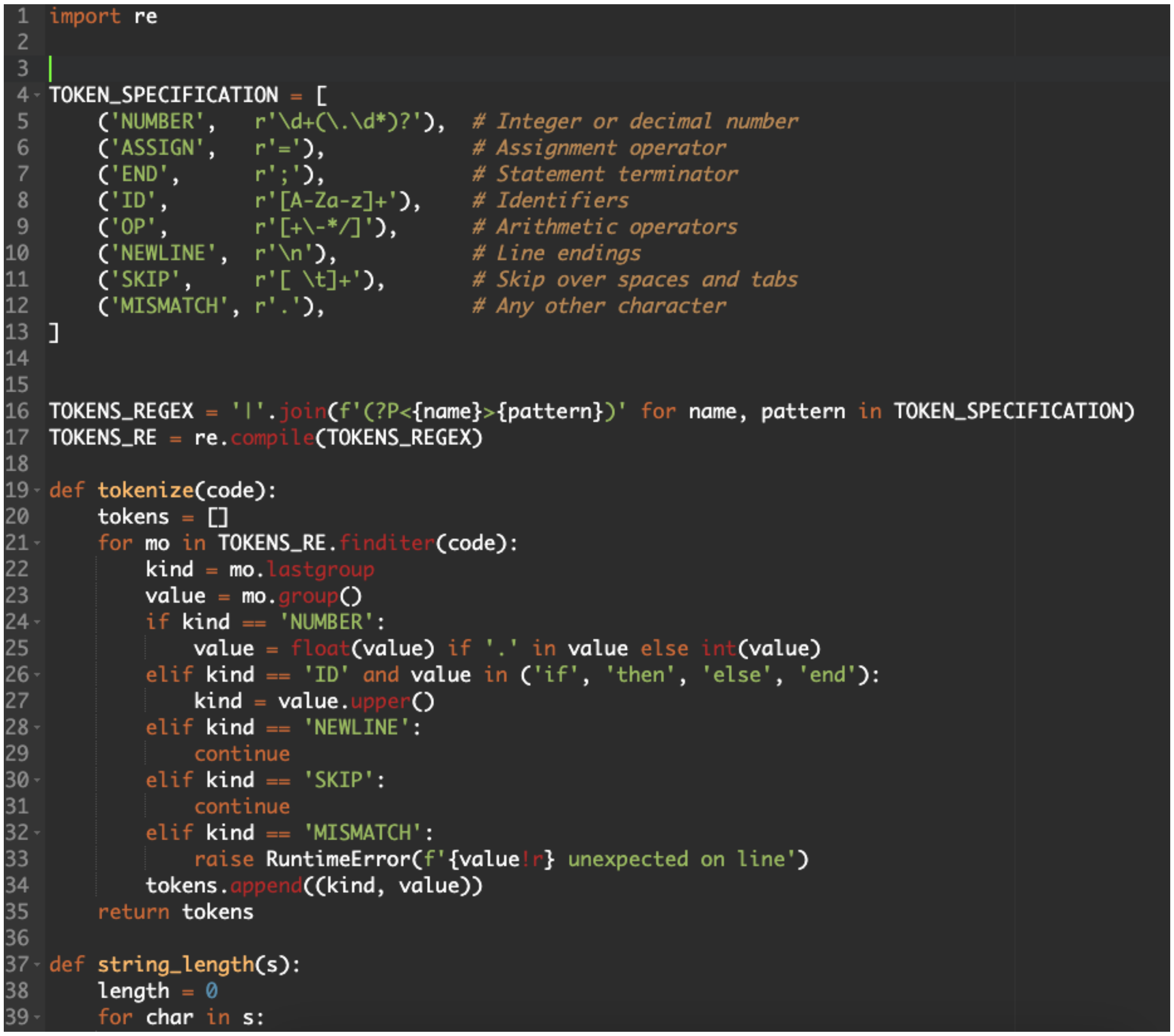
```python
def string_length(s):
    length = 0
    for char in s:
        length += 1
    return length

def string_copy(src):
    dest = ""
    for char in src:
        dest += char
    return dest

def string_uppercase(s):
    return s.upper()

def string_concatenate(s1, s2):
    return s1 + s2

def main():
    str1 = input("Enter the first string: ")
    str2 = input("Enter the second string: ")

    print(f"Length of '{str1}': {string_length(str1)}")
    print(f"Copied string: {string_copy(str1)}")
    print(f"Uppercase string: {string_uppercase(str1)}")
    print(f"Concatenated string: {string_concatenate(str1, str2)}")

if __name__ == "__main__":
    main()

```

Result:

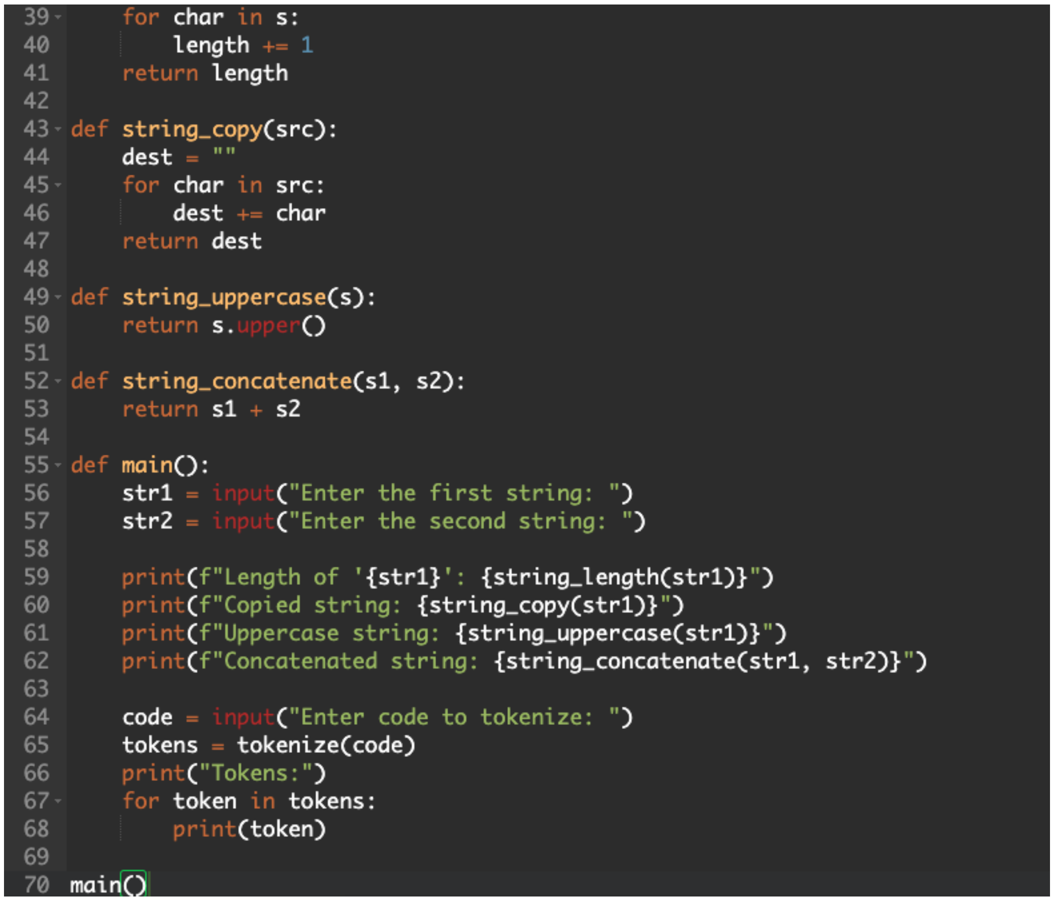


```
Enter the first string: Apurba
Enter the second string: Koirala
Length of 'Apurba': 6
Copied string: Apurba
Uppercase string: APURBA
Concatenated string: ApurbaKoirala
```

Token Specification in Python:

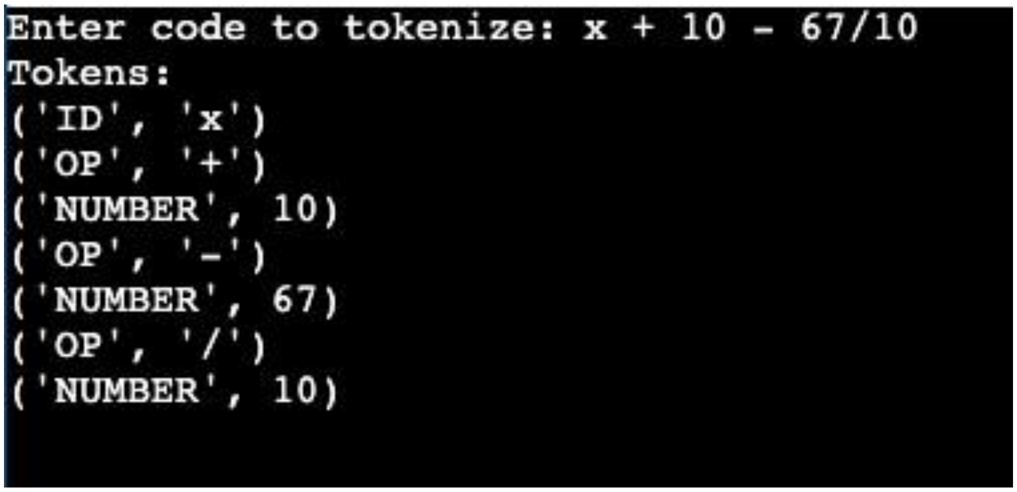


```python
import re


TOKEN_SPECIFICATION = [
    ('NUMBER',   r'\d+(\.\d*)?'),  # Integer or decimal number
    ('ASSIGN',   r'='),            # Assignment operator
    ('END',      r';'),            # Statement terminator
    ('ID',       r'[A-Za-z]+'),    # Identifiers
    ('OP',       r'[+\-*/]'),      # Arithmetic operators
    ('NEWLINE',  r'\n'),           # Line endings
    ('SKIP',     r'[ \t]+'),       # Skip over spaces and tabs
    ('MISMATCH', r'.'),            # Any other character
]


TOKENS_REGEX = '|'.join(f'(?P<{name}>{pattern})' for name, pattern in TOKEN_SPECIFICATION)
TOKENS_RE = re.compile(TOKENS_REGEX)

def tokenize(code):
    tokens = []
    for mo in TOKENS_RE.finditer(code):
        kind = mo.lastgroup
        value = mo.group()
        if kind == 'NUMBER':
            value = float(value) if '.' in value else int(value)
        elif kind == 'ID' and value in ('if', 'then', 'else', 'end'):
            kind = value.upper()
        elif kind == 'NEWLINE':
            continue
        elif kind == 'SKIP':
            continue
        elif kind == 'MISMATCH':
            raise RuntimeError(f'{value!r} unexpected on line')
        tokens.append((kind, value))
    return tokens

def string_length(s):
    length = 0
    for char in s:
```



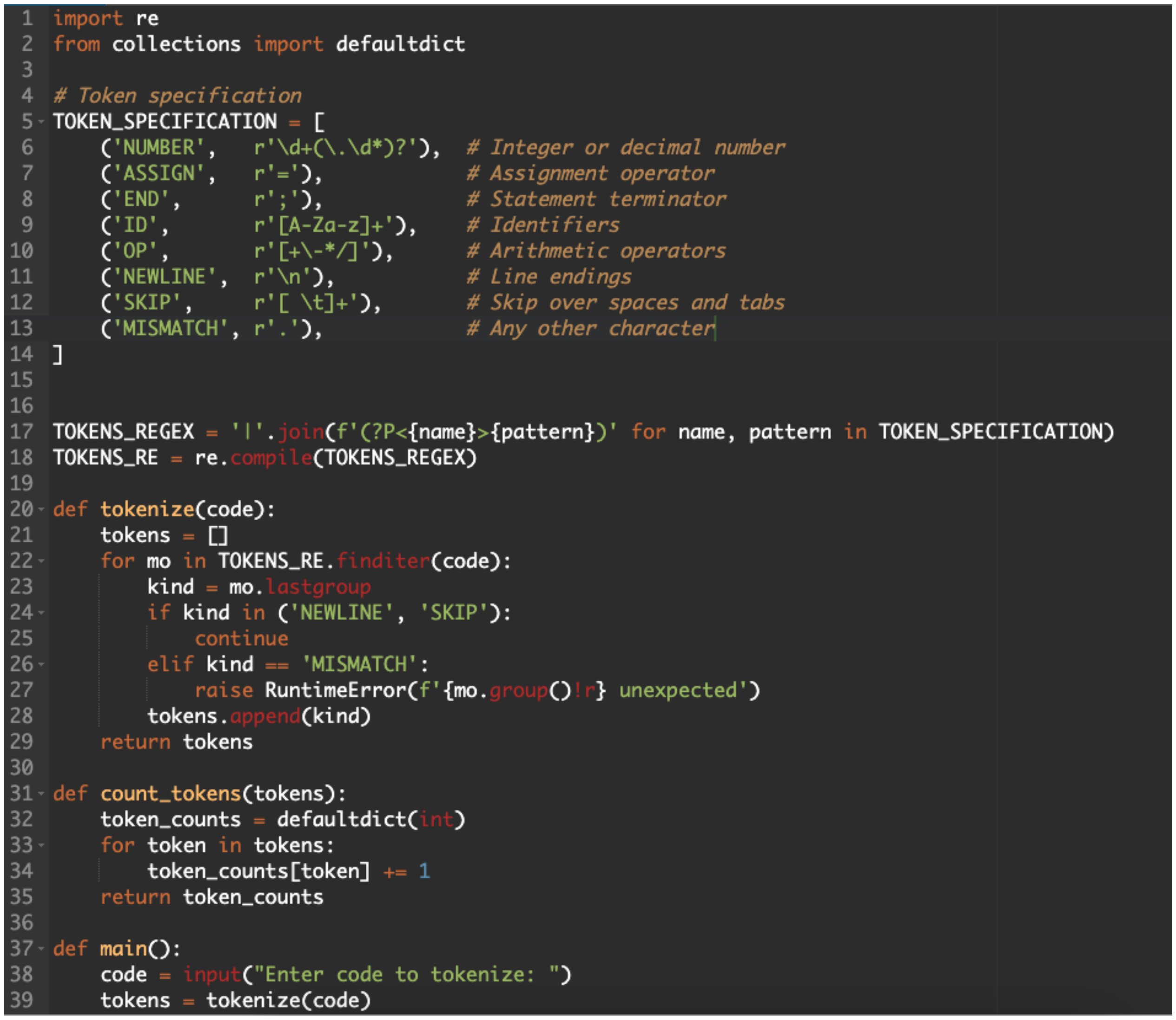
```python
    for char in s:
        length += 1
    return length

def string_copy(src):
    dest = ""
    for char in src:
        dest += char
    return dest

def string_uppercase(s):
    return s.upper()

def string_concatenate(s1, s2):
    return s1 + s2

def main():
    str1 = input("Enter the first string: ")
    str2 = input("Enter the second string: ")

    print(f"Length of '{str1}': {string_length(str1)}")
    print(f"Copied string: {string_copy(str1)}")
    print(f"Uppercase string: {string_uppercase(str1)}")
    print(f"Concatenated string: {string_concatenate(str1, str2)}")

    code = input("Enter code to tokenize: ")
    tokens = tokenize(code)
    print("Tokens:")
    for token in tokens:
        print(token)

main()
```

Result:

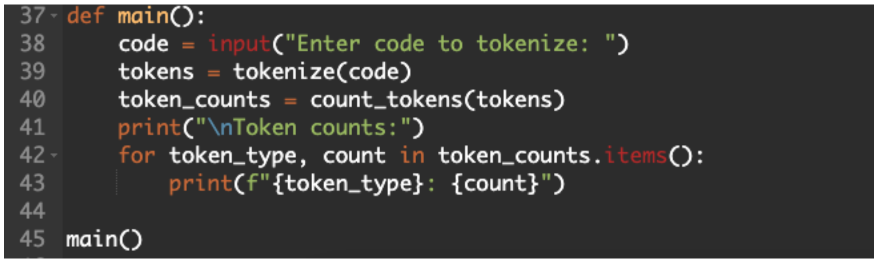


```
Enter code to tokenize: x + 10 - 67/10
Tokens:
('ID', 'x')
('OP', '+')
('NUMBER', 10)
('OP', '-')
('NUMBER', 67)
('OP', '/')
('NUMBER', 10)
```

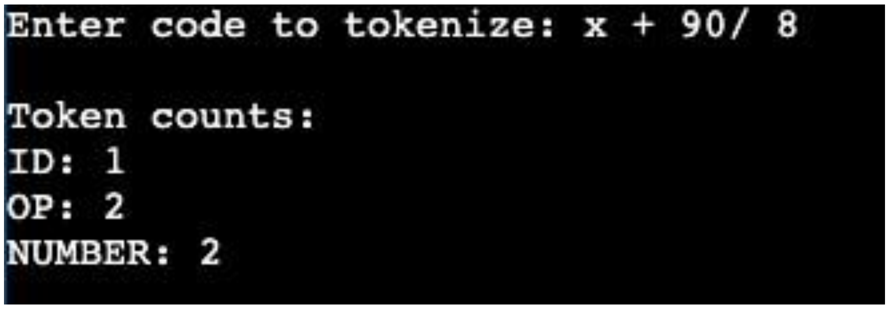
Token Count in Python:



```python
import re
from collections import defaultdict

# Token specification
TOKEN_SPECIFICATION = [
    ('NUMBER',   r'\d+(\.\d*)?'),  # Integer or decimal number
    ('ASSIGN',   r'='),            # Assignment operator
    ('END',      r';'),            # Statement terminator
    ('ID',       r'[A-Za-z]+'),    # Identifiers
    ('OP',       r'[+\-*/]'),      # Arithmetic operators
    ('NEWLINE',  r'\n'),           # Line endings
    ('SKIP',     r'[ \t]+'),       # Skip over spaces and tabs
    ('MISMATCH', r'.'),            # Any other character
]


TOKENS_REGEX = '|'.join(f'(?P<{name}>{pattern})' for name, pattern in TOKEN_SPECIFICATION)
TOKENS_RE = re.compile(TOKENS_REGEX)

def tokenize(code):
    tokens = []
    for mo in TOKENS_RE.finditer(code):
        kind = mo.lastgroup
        if kind in ('NEWLINE', 'SKIP'):
            continue
        elif kind == 'MISMATCH':
            raise RuntimeError(f'{mo.group()!r} unexpected')
        tokens.append(kind)
    return tokens

def count_tokens(tokens):
    token_counts = defaultdict(int)
    for token in tokens:
        token_counts[token] += 1
    return token_counts

def main():
    code = input("Enter code to tokenize: ")
    tokens = tokenize(code)
```



```python
def main():
    code = input("Enter code to tokenize: ")
    tokens = tokenize(code)
    token_counts = count_tokens(tokens)
    print("\nToken counts:")
    for token_type, count in token_counts.items():
        print(f"{token_type}: {count}")

main()
```



```
Enter code to tokenize: x + 90/ 8

Token counts:
ID: 1
OP: 2
NUMBER: 2
```