



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:

```
1- def string_length(s):
2-     length = 0
3-     for char in s:
4-         length += 1
5-     return length
6-
7- def string_copy(src):
8-     dest = ""
9-     for char in src:
10-        dest += char
11-     return dest
12-
13- def string_uppercase(s):
14-     return s.upper()
15-
16- def string_concatenate(s1, s2):
17-     return s1 + s2
18-
19- def main():
20-     str1 = input("Enter the first string: ")
21-     str2 = input("Enter the second string: ")
22-
23-     print(f"Length of '{str1}': {string_length(str1)}")
24-     print(f"Copied string: {string_copy(str1)}")
25-     print(f"Uppercase string: {string_uppercase(str1)}")
26-     print(f"Concatenated string: {string_concatenate(str1, str2)}")
27-
28- if __name__ == "__main__":
29-     main()
30- |
```

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:

```
1 import re
2
3
4 TOKEN_SPECIFICATION = [
5     ('NUMBER', r'\d+(\.\d*)?'), # Integer or decimal number
6     ('ASSIGN', r'='),           # Assignment operator
7     ('END', r';'),              # Statement terminator
8     ('ID', r'[A-Za-z]+'),       # Identifiers
9     ('OP', r'[+ \- * /]'),      # Arithmetic operators
10    ('NEWLINE', r'\n'),          # Line endings
11    ('SKIP', r'[ \t]+'),         # Skip over spaces and tabs
12    ('MISMATCH', r'.'),          # Any other character
13 ]
14
15
16 TOKENS_REGEX = '|'.join(f'(?P<{name}>{pattern})' for name, pattern in TOKEN_SPECIFICATION)
17 TOKENS_RE = re.compile(TOKENS_REGEX)
18
19 def tokenize(code):
20     tokens = []
21     for mo in TOKENS_RE.finditer(code):
22         kind = mo.lastgroup
23         value = mo.group()
24         if kind == 'NUMBER':
25             value = float(value) if '.' in value else int(value)
26         elif kind == 'ID' and value in ('if', 'then', 'else', 'end'):
27             kind = value.upper()
28         elif kind == 'NEWLINE':
29             continue
30         elif kind == 'SKIP':
31             continue
32         elif kind == 'MISMATCH':
33             raise RuntimeError(f'{value!r} unexpected on line')
34         tokens.append((kind, value))
35     return tokens
36
37 def string_length(s):
38     length = 0
39     for char in s:
```

```

39-     for char in s:
40-         length += 1
41-     return length
42-
43- def string_copy(src):
44-     dest = ""
45-     for char in src:
46-         dest += char
47-     return dest
48-
49- def string_uppercase(s):
50-     return s.upper()
51-
52- def string_concatenate(s1, s2):
53-     return s1 + s2
54-
55- def main():
56-     str1 = input("Enter the first string: ")
57-     str2 = input("Enter the second string: ")
58-
59-     print(f"Length of '{str1}': {string_length(str1)}")
60-     print(f"Copied string: {string_copy(str1)}")
61-     print(f"Uppercase string: {string_uppercase(str1)}")
62-     print(f"Concatenated string: {string_concatenate(str1, str2)}")
63-
64-     code = input("Enter code to tokenize: ")
65-     tokens = tokenize(code)
66-     print("Tokens:")
67-     for token in tokens:
68-         print(token)
69-
70- 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:

```
1 import re
2 from collections import defaultdict
3
4 # Token specification
5 TOKEN_SPECIFICATION = [
6     ('NUMBER', r'\d+(\.\d*)?'), # Integer or decimal number
7     ('ASSIGN', r'='),           # Assignment operator
8     ('END', r';'),              # Statement terminator
9     ('ID', r'[A-Za-z]+'),       # Identifiers
10    ('OP', r'[+\-*/]'),          # Arithmetic operators
11    ('NEWLINE', r'\n'),          # Line endings
12    ('SKIP', r'[ \t]+'),         # Skip over spaces and tabs
13    ('MISMATCH', r'.'),          # Any other character
14 ]
15
16
17 TOKENS_REGEX = '|'.join(f'(?P<{name}>{pattern})' for name, pattern in TOKEN_SPECIFICATION)
18 TOKENS_RE = re.compile(TOKENS_REGEX)
19
20 def tokenize(code):
21     tokens = []
22     for mo in TOKENS_RE.finditer(code):
23         kind = mo.lastgroup
24         if kind in ('NEWLINE', 'SKIP'):
25             continue
26         elif kind == 'MISMATCH':
27             raise RuntimeError(f'{mo.group()!r} unexpected')
28         tokens.append(kind)
29     return tokens
30
31 def count_tokens(tokens):
32     token_counts = defaultdict(int)
33     for token in tokens:
34         token_counts[token] += 1
35     return token_counts
36
37 def main():
38     code = input("Enter code to tokenize: ")
39     tokens = tokenize(code)
```

```
37 def main():
38     code = input("Enter code to tokenize: ")
39     tokens = tokenize(code)
40     token_counts = count_tokens(tokens)
41     print("\nToken counts:")
42     for token_type, count in token_counts.items():
43         print(f"{token_type}: {count}")
44
45 main()
```

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

Token counts:

ID: 1

OP: 2

NUMBER: 2