EXPERIMENT NO. - 02

AIM: Edit/compile/run a program to display the statements on two different lines.

THEORY:

Many scripting and programming languages, such as JScript, C#, and C++, make no attempt to match the code that is run with the actual physical lines typed into the text editor. This is because they not recognize the end of a line of code until it sees the termination character (in these cases, the semicolon). Thus, the actual physical lines of type taken up by the code are irrelevant.

Unlike other languages, Python does not use an end of line character. Most of the time a simple Enter will do. Yet, Python is very particular about indentation, spaces and lines in certain cases. This document is to help understand Python formatting.

It is important to understand how Python interprets:

2. End of Statement
3. Names and Capitalization
4. Comments
5. Block Structures
6. Tabs and Spaces

End of Statements

To end a statement in Python, you do not have to type in a semicolon or other special character; you simply press Enter. For example, this code will generate a syntax error:

```
message

=

'Hello World!'
```

This will not:

```
message = 'Hello World!'
```

In general, the lack of a required statement termination character simplifies script writing in Python. There is, however, one complication: To enhance readability, it is recommended that you limit the length of any single line of code to 79 characters. What happens, then, if you have a line of code that contains 100 characters?

Although it might seem like the obvious solution, you cannot split a statement into multiple lines simply by entering a carriage return. For example, the following code snippet returns a run-time error in Python because a statement was split by using Enter.

```
message= 'This message will generate an error because

it was split by using the enter button on your

keyboard'
```

You cannot split a statement into multiple lines in Python by pressing Enter. Instead, use the backslash (\) to indicate that a statement is continued on the next line. In the revised version of the script, a blank space and an underscore indicate that the statement that was started on line 1 is continued on line 2. To make it more apparent that line 2 is a continuation of line 1, line 2 is also indented four spaces. (This was done for the sake of readability, but you do not have to indent continued lines.)

```
message\
=\
'This \
back slash \
acts \
like \
enter'
print\
message
message\
=\
“""triple
quotes
will
span
multiple lines
without
errors"””
print\
message
```

Line continuation is automatic when the split comes while a statement is inside parenthesis (( ), brackets ([ ) or braces ({). This is convenient, but can also lead to errors if there is no closing Parenthesis, bracket or brace. Python would interpret the rest of the script as one statement in that case.

Python uses single quotes (') double quotes (") and triple quotes (""") to denote literal strings. Only the triple quoted strings (""") also will automatically continue across the end of line statement.

Sometimes, more than one statement may be put on a single line. In Python a semicolon (;) can be used to separate multiple statements on the same line. For instance three statements can be written:

```
y = 3; x = 5; print(x+y)
```

To the Python interpreter, this would be the same set of statements:

```
y = 3
x = 5
print(x+y)
```

EXPERIMENT NO. - 02

PROGRAM:

EXPT 2) A:

```
print('Hello World!')
print('Welcome to Python Programming!')

print(""" This is the Strangest
way to print over
     multiple lines I know!""")
```

EXPT 2) B:

```
# Python multiline string with newlines example using brackets
multiline_str = ("Hello World! \n"
"Welcome to Python Programming! \n"
"I'm learning Python.\n"
"I refer to Lectures, Notes & Tutorials given by \n Subject Teacher: Prof. K. R. Korpe.\n"
"If you want a line break in the STRING, then you can use" "\n" " as a string literal wherever you
need it.")
print(multiline_str)
```