**Assignment-17**

**Q1. Explain the difference between greedy and non-greedy syntax with visual terms in as few words as possible. What is the bare minimum effort required to transform a greedy pattern into a non-greedy one? What characters or characters can you introduce or change?**

Example: Greedy pattern matching

#!/usr/bin/perl

# Perl program to show greedy matching

$var = "Geeks For Geeks";

# Matching pattern from k to s

$var =~ /(k.\*s)(.\*)$/;

# Printing the resultant string

print($1, "\n");

Example: Non-greedy pattern matching

#!/usr/bin/perl

# Perl program to show non-greedy matching

my $str = "Geeks For Geeks";

# Matching pattern from k to s

$str =~ /.\*?\s(.\*)/;

# Printing Resultant string

print($1);

**Q2. When exactly does greedy versus non-greedy make a difference?  What if you're looking for a non-greedy match but the only one available is greedy?**

In greedy approach regex pattern tend to consume maximum characters in a source string.For example

textstr = "bcabdcab"

textstr.gsub!(/(.\*)ab/, "xxx")

# this will match whole `bcabdcab` and return `xxx`

Here \* is a greedy quantifier.In non greedy approach regex engine returns when it satisfies the matching criteria.To make a quantifier non-greedy append ?

textstr = "bcabdcab"

textstr.gsub!(/(.\*?)ab/, "xxx")

# this will match only `bcab` part and return `xxxdcab`

**Q3. In a simple match of a string, which looks only for one match and does not do any replacement, is the use of a nontagged group likely to make any practical difference?**

Regular expressions are useful in any scenario that benefits from full or partial pattern matches on strings. These are some common use cases:

verify the structure of strings

extract substrings form structured strings

search / replace / rearrange parts of the string

split a string into tokens

**Q4. Describe a scenario in which using a nontagged category would have a significant impact on the program's outcomes.**

Workflow forms appear under the tag name in the Web Console. If you leave the Tags property empty for a form, the form appears under the Untagged category. If none of the workflow forms have tags, the forms appear without grouping. For information about workflow forms, see

**Q5. Unlike a normal regex pattern, a look-ahead condition does not consume the characters it examines. Describe a situation in which this could make a difference in the results of your programme.**

# importing regex

import re

# lookahead example

example = re.search(r'geeks(?=[a-z])', "geeksforgeeks")

# display output

print("Pattern:", example.group())

print("Pattern found from index:",

example.start(), "to",

example.end())

**Q6. In standard expressions, what is the difference between positive look-ahead and negative look-ahead?**

# import required module

import re

# positive lookahead

example1 = re.search('geeks(?=[a-z])',

'geeksforgeeks')

print('Positive Lookahead:', example1.group())

# negative lookahead

example2 = re.search('geeks(?![a-z])',

'geeks123')

print('Negative Lookahead:', example2.group())

**Q7. What is the benefit of referring to groups by name rather than by number in a standard expression?**

This is followed by an optional character and a space. The third named group is year. This consists of 1 or more digits. The advantage to named groups is that it adds readability and understandability to the code, so that you can easily see what part of a regular expression match is being referenced.

**Q8. Can you identify repeated items within a target string using named groups, as in "The cow jumped over the moon"?**

using std::regex;

using std::regex\_replace;

using std::string;

regex reg1("([A-Za-z]+) \\1"); // Find double word.

string replacement = "$1"; // Replace with one word.

string target = "The cow cow jumped over the the moon.";

string result = regex\_replace(target, reg1, replacement);

std::cout << result << std::endl;

**Q9. When parsing a string, what is at least one thing that the Scanner interface does for you that the re.findall feature does not?**

findall() Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found.

**Q10. Does a scanner object have to be named scanner?**

**Ans:** No