1. **Characters which are used to match starts with and ends with?**

**Ans**: starts with: ^

Ends with: $

1. **List of modifiers in Regular Expressions**

**Ans:**

|  |  |
| --- | --- |
| 1 | **re.I**  Performs case-insensitive matching. |
| 2 | **re.L**  Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B). |
| 3 | **re.M**  Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string). |
| 4 | **re.S**  Makes a period (dot) match any character, including a newline. |
| 5 | **re.U**  Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B. |
| 6 | **re.X**  Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker. |

1. **Write a Regular expression to match IP Address, Phone Number and Email id?**

**Ans**:

**For IP Address:**

ip\_address\_regex\_= re.compile("^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$")

**For Phone Number:**

phone\_number\_regex\_pattern = r"\(?\d{3}\)?[-.\s]\d{3}[-.\s]\d{4}"

**For Email ID:**

Email\_id\_regex\_pattern = r'[\w\.-][+@[\w\.-]+(\.[\w]+)](mailto:+@[\w\.-%5d+(\.%5b\w%5d+))+'

1. **Write Regular Expression to match all three letter word which ends with ”at” for following :**

**Ans**:

re.findall(r'\b([a-zA-Z]{1}at)\b',paragraph)

1. **Explain the following and give some examples,**

**Ans**:

**Matching .{4} Repetitions** **:** Match any word or number of length 4

**Matching \D{1,5} Repetitions** **:** Matches any non-decimal digit. Equivalent to [^0-9] of word length 1 to 5

**Matching Zero or more Repetitions** **:** {0,10}

**Matching One or more Repetitions** **:** {1,10}

1. **Write a Regular expression to match the following.**

**Ans**:

re.findall(r'\b[A-Z]{1}[a-z]{1,2}\.',s)

1. **Explain capturing and Non-capturing group and give some examples?**

**Ans**:

**Capturing:**

Parentheses group the regex between them. They capture the text matched by the regex inside them into a numbered group that can be reused with anumbered backreference. They allow you to apply regex operators to the entire grouped regex.

Syntax: (regex)

Ex:

import re

txt = """ i love cats

i love dogs"""

pattern = re.compile("i love (cats|dogs)")

pattern.findall(txt)

for match in pattern.finditer(txt):

    print("complete regex match (default)",match.group(0))

    print("match captured by 1 st group:", match.group(1))

**output**:

complete regex match (default) i love cats

match captured by 1 st group: cats

complete regex match (default) i love dogs

match captured by 1 st group: dogs

here caputured contains only cats or dogs instead of complete sentences

**Non-capturing group:**

Non caputuring groups are when we want to use groups but we’re not interested in extracting information .i.e. caputuring the matched text inside parenthesis only an example :

**Syntax**: (?:pattern)

Ex:

import re

txt = """ i like cats  i like dogs"""

pattern = re.compile("i  like (?:cats|dogs)")

pattern.findall(txt)

for match in pattern.finditer(txt):

    print(match.groups(0))

**output**:

i like cats

i like dogs

1. **Write a regular expression to match a complete HTML source page.(use:** [**https://www.google.com.in/**](https://www.google.com.in/)**)**

**Ans:**

regex =r"(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a- z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))\*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))\*\)|[^\s`!()\[\]{};:'\".,<>?«»“”‘’]))"

url = re.findall(regex,string)

result = [x[0] for x in url]

1. **Write a regular expressions to match alphanumeric word which should not match any special characters?**

**Ans**:

re.findall(r'\w+',s)

1. **Explain the following:**

**Ans:**

1. \d **:**  Matches any decimal digit. Equivalent to [0-9]
2. \D+**:** Matches any non-decimal digit. Equivalent to [^0-9] one or more
3. \w **:** Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9\_]. By the way, underscore \_ is also considered an alphanumeric
4. .+ **:** one or more full stops.
5. \S+ **:** Matches where a string contains any non-whitespace character One or more.