Regular Expression

1. Regular expression is a powerful tool for matching text-based on pre-defined pattern
2. Regular expression can detect the presence of text by matching with perticular pattern,
3. also split into number of sub parameter
4. It represent the group of string according to perticular pattern then we should go for Regular Expression
5. Regular expression are widely used in UNIX world
6. Regular expression is powerful language for matching pattern and text pattern
7. Regular expression is sequnce of characters that are used to create search pattern
8. Python has built in package called as [re]
9. Regex can be used check if a string contain the specified search pattern
10. re module have different types of functions which allowed us to search string match.:
    A. Search
    B. Findall
    C. Split
    D. Sub

Metacharacters

1. [] - Use to create a Set of Characters
2. [.] - Matches any character except a newline
3. [] - Either escapes special characters or signals a special sequence
4. [^] - Starts with
5. [$] - Ends with
6. [*] - Zero or More Occurance
7. [+] - One or Occurance
8. {} - Exactly the specified number of occurances
9. | - Either or
10. ()- Capture Group

In [18]:
```
import re
```

In [31]:
```
my_text = "My name is Python, My email id is python@gmail.com."
pattern = re.search("^My", my_text)
```

In [32]:
```
pattern
```

Out[32]:
```
<re.Match object; span=(0, 2), match='My'>
```

In [23]:
```
my_text = "My name is Python, My email id python@gmail.com are"
pattern = re.search("are$", my_text)
```

In [24]:
```
pattern
```

Out[24]:
```
<re.Match object; span=(48, 51), match='are'>
```

In [52]:
```
my_text = "My name is Python, My email id is python@gmail.com. Hello"
pattern = re.search("^.*", my_text)
```

In [53]:
```
pattern
```

Out[53]:
```
<re.Match object; span=(0, 57), match='My name is Python, My email id is pyth
on@gmail.co>
```

In [54]:
```
help(re)
```

```
Help on module re:

NAME
    re - Support for regular expressions (RE).

MODULE REFERENCE
    https://docs.python.org/3.8/library/re

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides regular expression matching operations similar to
    those found in Perl.  It supports both 8-bit and Unicode strings; both
    the pattern and the strings being processed can contain null bytes and
    characters outside the US ASCII range.

    Regular expressions can contain both special and ordinary characters.
    Most ordinary characters, like "A", "a", or "0", are the simplest
    regular expressions; they simply match themselves.  You can
    concatenate ordinary characters, so last matches the string 'last'.

    The special characters are:
        "."      Matches any character except a newline.
        "^"      Matches the start of the string.
        "$"      Matches the end of the string or just before the newline at
                 the end of the string.
        "*"      Matches 0 or more (greedy) repetitions of the preceding RE.
                 Greedy means that it will match as many repetitions as possi
ble.
        "+"      Matches 1 or more (greedy) repetitions of the preceding RE.
        "?"      Matches 0 or 1 (greedy) of the preceding RE.
        *?,+?,?? Non-greedy versions of the previous three special character
s.
        {m,n}    Matches from m to n repetitions of the preceding RE.
        {m,n}?   Non-greedy version of the above.
        "\\"     Either escapes special characters or signals a special seque
nce.
        []       Indicates a set of characters.
                 A "^" as the first character indicates a complementing set.
        "|"      A|B, creates an RE that will match either A or B.
        (...)    Matches the RE inside the parentheses.
                 The contents can be retrieved or matched later in the strin
g.
        (?aiLmsux) The letters set the corresponding flags defined below.
        (?:...)  Non-grouping version of regular parentheses.
        (?P<name>...) The substring matched by the group is accessible by nam
e.
        (?P=name)     Matches the text matched earlier by the group named nam
e.
        (?#...)  A comment; ignored.
        (?=...)  Matches if ... matches next, but doesn't consume the string.
        (?!...)  Matches if ... doesn't match next.
        (?<=...) Matches if preceded by ... (must be fixed length).
```

```
        (?<!...) Matches if not preceded by ... (must be fixed length).
        (?(id/name)yes|no) Matches yes pattern if the group with id/name matc
hed,
                           the (optional) no pattern otherwise.

    The special sequences consist of "\\" and a character from the list
    below.  If the ordinary character is not on the list, then the
    resulting RE will match the second character.
        \number  Matches the contents of the group of the same number.
        \A       Matches only at the start of the string.
        \Z       Matches only at the end of the string.
        \b       Matches the empty string, but only at the start or end of a
word.
        \B       Matches the empty string, but not at the start or end of a w
ord.
        \d       Matches any decimal digit; equivalent to the set [0-9] in
                 bytes patterns or string patterns with the ASCII flag.
                 In string patterns without the ASCII flag, it will match the
whole
                 range of Unicode digits.
        \D       Matches any non-digit character; equivalent to [^\d].
        \s       Matches any whitespace character; equivalent to [ \t\n\r\f
\v] in
                 bytes patterns or string patterns with the ASCII flag.
                 In string patterns without the ASCII flag, it will match the
whole
                 range of Unicode whitespace characters.
        \S       Matches any non-whitespace character; equivalent to [^\s].
        \w       Matches any alphanumeric character; equivalent to [a-zA-Z0-9
_]
                 in bytes patterns or string patterns with the ASCII flag.
                 In string patterns without the ASCII flag, it will match the
                 range of Unicode alphanumeric characters (letters plus digit
s
                 plus underscore).
                 With LOCALE, it will match the set [0-9_] plus characters de
fined
                 as letters for the current locale.
        \W       Matches the complement of \w.
        \\       Matches a literal backslash.

    This module exports the following functions:
        match     Match a regular expression pattern to the beginning of a st
ring.
        fullmatch Match a regular expression pattern to all of a string.
        search    Search a string for the presence of a pattern.
        sub       Substitute occurrences of a pattern found in a string.
        subn      Same as sub, but also return the number of substitutions ma
de.
        split     Split a string by the occurrences of a pattern.
        findall   Find all occurrences of a pattern in a string.
        finditer  Return an iterator yielding a Match object for each match.
        compile   Compile a pattern into a Pattern object.
        purge     Clear the regular expression cache.
        escape    Backslash all non-alphanumerics in a string.

    Each function other than purge and escape can take an optional 'flags' ar
```

```
gument
    consisting of one or more of the following module constants, joined by
"|".
    A, L, and U are mutually exclusive.
        A  ASCII       For string patterns, make \w, \W, \b, \B, \d, \D
                       match the corresponding ASCII character categories
                       (rather than the whole Unicode categories, which is th
e
                       default).
                       For bytes patterns, this flag is the only available
                       behaviour and needn't be specified.
        I  IGNORECASE  Perform case-insensitive matching.
        L  LOCALE      Make \w, \W, \b, \B, dependent on the current locale.
        M  MULTILINE   "^" matches the beginning of lines (after a newline)
                       as well as the string.
                       "$" matches the end of lines (before a newline) as wel
l
                       as the end of the string.
        S  DOTALL      "." matches any character at all, including the newlin
e.
        X  VERBOSE     Ignore whitespace and comments for nicer looking RE's.
        U  UNICODE     For compatibility only. Ignored for string patterns (i
t
                       is the default), and forbidden for bytes patterns.

    This module also defines an exception 'error'.

CLASSES
    builtins.Exception(builtins.BaseException)
        error
    builtins.object
        Match
        Pattern

    class Match(builtins.object)
     |  The result of re.match() and re.search().
     |  Match objects always have a boolean value of True.
     |
     |  Methods defined here:
     |
     |  __copy__(self, /)
     |
     |  __deepcopy__(self, memo, /)
     |
     |  __getitem__(self, key, /)
     |      Return self[key].
     |
     |  __repr__(self, /)
     |      Return repr(self).
     |
     |  end(self, group=0, /)
     |      Return index of the end of the substring matched by group.
     |
     |  expand(self, /, template)
     |      Return the string obtained by doing backslash substitution on the
string template, as done by the sub() method.
     |
```

```
 |  group(...)
 |      group([group1, ...]) -> str or tuple.
 |      Return subgroup(s) of the match by indices or names.
 |      For 0 returns the entire match.
 |
 |  groupdict(self, /, default=None)
 |      Return a dictionary containing all the named subgroups of the mat
ch, keyed by the subgroup name.
 |
 |      default
 |        Is used for groups that did not participate in the match.
 |
 |  groups(self, /, default=None)
 |      Return a tuple containing all the subgroups of the match, from 1.
 |
 |      default
 |        Is used for groups that did not participate in the match.
 |
 |  span(self, group=0, /)
 |      For match object m, return the 2-tuple (m.start(group), m.end(gro
up)).
 |
 |  start(self, group=0, /)
 |      Return index of the start of the substring matched by group.
 |
 |  ----------------------------------------------------------------------
-
 |  Data descriptors defined here:
 |
 |  endpos
 |      The index into the string beyond which the RE engine will not go.
 |
 |  lastgroup
 |      The name of the last matched capturing group.
 |
 |  lastindex
 |      The integer index of the last matched capturing group.
 |
 |  pos
 |      The index into the string at which the RE engine started looking
for a match.
 |
 |  re
 |      The regular expression object.
 |
 |  regs
 |
 |  string
 |      The string passed to match() or search().

class Pattern(builtins.object)
 |  Compiled regular expression object.
 |
 |  Methods defined here:
 |
 |  __copy__(self, /)
 |
```

```
 |  __deepcopy__(self, memo, /)
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  findall(self, /, string, pos=0, endpos=9223372036854775807)
 |      Return a list of all non-overlapping matches of pattern in strin
g.
 |
 |  finditer(self, /, string, pos=0, endpos=9223372036854775807)
 |      Return an iterator over all non-overlapping matches for the RE pa
ttern in string.
 |
 |      For each match, the iterator returns a match object.
 |
 |  fullmatch(self, /, string, pos=0, endpos=9223372036854775807)
 |      Matches against all of the string.
 |
 |  match(self, /, string, pos=0, endpos=9223372036854775807)
 |      Matches zero or more characters at the beginning of the string.
 |
 |  scanner(self, /, string, pos=0, endpos=9223372036854775807)
 |
 |  search(self, /, string, pos=0, endpos=9223372036854775807)
 |      Scan through string looking for a match, and return a correspondi
ng match object instance.
 |
 |      Return None if no position in the string matches.
 |
 |  split(self, /, string, maxsplit=0)
 |      Split string by the occurrences of pattern.
 |
 |  sub(self, /, repl, string, count=0)
 |      Return the string obtained by replacing the leftmost non-overlapp
ing occurrences of pattern in string by the replacement repl.
 |
```

```
 |  subn(self, /, repl, string, count=0)
 |      Return the tuple (new_string, number_of_subs_made) found by repla
cing the leftmost non-overlapping occurrences of pattern with the replacement
repl.
 |
 |  ----------------------------------------------------------------------
-
 |  Data descriptors defined here:
 |
 |  flags
 |      The regex matching flags.
 |
 |  groupindex
 |      A dictionary mapping group names to group numbers.
 |
 |  groups
 |      The number of capturing groups in the pattern.
 |
 |  pattern
 |      The pattern string from which the RE object was compiled.

class error(builtins.Exception)
 |  error(msg, pattern=None, pos=None)
 |
 |  Exception raised for invalid regular expressions.
 |
 |  Attributes:
 |
 |      msg: The unformatted error message
 |      pattern: The regular expression pattern
 |      pos: The index in the pattern where compilation failed (may be No
ne)
 |      lineno: The line corresponding to pos (may be None)
 |      colno: The column corresponding to pos (may be None)
 |
 |  Method resolution order:
 |      error
 |      builtins.Exception
 |      builtins.BaseException
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, msg, pattern=None, pos=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
-
 |  Data descriptors defined here:
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
-
 |  Static methods inherited from builtins.Exception:
 |
```

```
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate sign
ature.
 |
 |  ----------------------------------------------------------------------
-
 |  Methods inherited from builtins.BaseException:
 |
 |  __delattr__(self, name, /)
 |      Implement delattr(self, name).
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(...)
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  with_traceback(...)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
-
 |  Data descriptors inherited from builtins.BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

FUNCTIONS
    compile(pattern, flags=0)
        Compile a regular expression pattern, returning a Pattern object.

    escape(pattern)
        Escape special characters in a string.
```

```
findall(pattern, string, flags=0)
    Return a list of all non-overlapping matches in the string.

    If one or more capturing groups are present in the pattern, return
    a list of groups; this will be a list of tuples if the pattern
    has more than one group.

    Empty matches are included in the result.

finditer(pattern, string, flags=0)
    Return an iterator over all non-overlapping matches in the
    string.  For each match, the iterator returns a Match object.

    Empty matches are included in the result.

fullmatch(pattern, string, flags=0)
    Try to apply the pattern to all of the string, returning
    a Match object, or None if no match was found.

match(pattern, string, flags=0)
    Try to apply the pattern at the start of the string, returning
    a Match object, or None if no match was found.

purge()
    Clear the regular expression caches

search(pattern, string, flags=0)
    Scan through string looking for a match to the pattern, returning
    a Match object, or None if no match was found.

split(pattern, string, maxsplit=0, flags=0)
    Split the source string by the occurrences of the pattern,
    returning a list containing the resulting substrings.  If
    capturing parentheses are used in pattern, then the text of all
    groups in the pattern are also returned as part of the resulting
    list.  If maxsplit is nonzero, at most maxsplit splits occur,
    and the remainder of the string is returned as the final element
    of the list.

sub(pattern, repl, string, count=0, flags=0)
    Return the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in string by the
    replacement repl.  repl can be either a string or a callable;
    if a string, backslash escapes in it are processed.  If it is
    a callable, it's passed the Match object and must return
    a replacement string to be used.

subn(pattern, repl, string, count=0, flags=0)
    Return a 2-tuple containing (new_string, number).
    new_string is the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in the source
    string by the replacement repl.  number is the number of
    substitutions that were made. repl can be either a string or a
    callable; if a string, backslash escapes in it are processed.
    If it is a callable, it's passed the Match object and must
    return a replacement string to be used.
```

```
        template(pattern, flags=0)
            Compile a template pattern, returning a Pattern object

    DATA
        A = re.ASCII
        ASCII = re.ASCII
        DOTALL = re.DOTALL
        I = re.IGNORECASE
        IGNORECASE = re.IGNORECASE
        L = re.LOCALE
        LOCALE = re.LOCALE
        M = re.MULTILINE
        MULTILINE = re.MULTILINE
        S = re.DOTALL
        U = re.UNICODE
        UNICODE = re.UNICODE
        VERBOSE = re.VERBOSE
        X = re.VERBOSE
        __all__ = ['match', 'fullmatch', 'search', 'sub', 'subn', 'split', 'fi...

    VERSION
        2.2.1

    FILE
        c:\users\shyam.desktop-3em50g2\anaconda3\lib\re.py
```

In [ ]: